Implementation of Improved EDC Combustion Model in the Open LES Code FDS

David R.U. Johansen

Master's thesis in Process Safety
University of Bergen
Department of Physics and Technology
Bergen, Norway

August 2011

Preface

This thesis is submitted as the final part of the degree Master of Science in Process Safety Engineering at the University of Bergen. The work has been carried out at Stord/Haugesund University College throughout the whole period. To work with this thesis has been both challenging and time consuming. Still, it has been a valuable experience and really exciting.

The major challenge of this thesis has not been to implement the code in FDS, but to adapt it to the rest of the code and to orientate in the in the jungle of modules and subroutines. To handle the large amount of numbers requires systematic work and has also been challenging. Unfortunately, the experiments at Lund University gave unsuccessful PIV data. However, the main goal with the experiments was to gain more knowledge to experimental work and not validate the implemented code.

To be able to quantify such complex physical and chemical phenomenon as a fire, and also fluid flow in general with CFD, is truly fascinating me. Since the use of CFD requires a broad knowledge in scientific topics makes it extra interesting. Through this project I have linked loose ends of knowledge together from earlier courses and learned how to apply it in the context of CFD. I have also gained more knowledge to numerical methods and experimental work. However, it is still much more to learn. To work with this thesis has encouraged me to continue to work in the field of fire research.

Acknowlegdements

I would like to express my sincere gratitude to my supervisor Associate Professor Bjarne P. Husted at Stord/Haugesund University College. He has been a great inspiration and encouraging to work with. Throughout the process of my work, Bjarne P. Husted has been providing me with expert advice on CFD, fire dynamics, computer science, numerical methods and experimental work. I really appreciate that he linked me up to the pool fire project and even joined the experiments at Lund University.

I appreciate that the staff at Lund University let me take part of the pool fire project, and at the same time let me to do experiments relating my Master's thesis. I would like to particularly thank Mr. Per Petersson for operating PIV measurements, Professor Patrick van Hees for leading the experiments and PhD student Jonathan Wahlquist who helped with set up.

Other people I would like to thank are:
- Dr. Balram Panjwani (Norwegian University of Science and Technology) who e-mailed me his PhD thesis and answered questions regarding LES-EDC
- Ms. Mari Ueland and Ms. Siri H. Walaker for feedbacks on my report
- my older brother Mr. Remi Johansen who helped me with Linux, MatLab and $\LaTeX$
- Dr. Monika Metallinou who arranged an office for me at Stord/Haugesund University College
- Professor Bjørn Arntzen at the University of Bergen and fellow Master student Ms. Natalja Pedersen for discussion of EDC
- PhD student Bjarne Chr. Hagen (Stord/Haugesund University College) and fellow Master student Mr. Einar Kolstad for general discussion and advice
- Dr. Alf Reidar Nilsen (Stord/Haugesund University College and Statoil) for letting me use his computers for simulations
- Mr. Arjen Kraaijeveld (Stord/Haugesund University College) for advices regarding experimental work

Summary

In this thesis, Magnussen's and Hjertager's Eddy Dissipation Concept (EDC) Combustion Model [1][2][3] has been implemented in the CFD code Fire Dynamics Simulator (FDS) [4]. FDS is an open source non-commercial Large Eddy Simulation (LES) code mainly developed by the National Institute of Standards and Technology (NIST). EDC is developed for Reynolds Avarage Navier Stokes (RANS) equations where fluctuating values caused by turbulence are modeled. The reaction rate in EDC is predicted by RANS quantities such as Turbulent Kinetic Energy (TKE) and Dissipation of Turbulent Kinetic Energy, which is not solved explicitly in LES codes. A promising extension of EDC to LES was proposed Panjwani *et al.* treating eddy viscosity and strain rate instead [5][6]. In their validation work a model constant was established for jet flames for the Smagorinsky turbulence model. However, the main purpose of FDS is studying smoke spread, fire detection and smoke ventilation in building fires. Such fires involves low Mach number flows driven by buoyancy, in contrast to jet fire that are strongly influenced by the momentum fuel release and are highly turbulent.

The first motivation of this thesis has been to implement LES-EDC in FDS, and second to establish a model constant for buoyancy driven fires and evaluate whether a static constant is sufficient or a dynamic constant is necessary. This is solved by validate the code against velocity profiles in Sandia plume experiments, Heskestad flame height correlation, McCaffery centerline temperature and velocity correlation. Results are also compared with the existing combustion model in FDS for the default turbulence model, Deardorff, in the unofficial version 6.

Experiments with square pipes inserted in the persistent flame region were performed at Lund University. Particle Image Velocimetry (PIV) technique was applied to measure the velocity vector field above the pipes. The goal was to study the affected of generated turbulence from the pipes and perhaps be able to investigate how the turbulence effected the LES-EDC model constant. Unfortunately, the experiments gave unsatisfactory results for CFD validation. During the experiments it was some technical problems with the shutter on the camera occurred as well as the seeding of particles turned out to be quite challenging. Therefore, this part of the thesis must be regarded as a contribution to the project *Prediction and validation of pool fire developed in enclosures by means of CFD models for risk assessment of nuclear power plants* which the experiments were linked up to, and not validation of FDS-EDC.

A model constant of 0.015 gave satisfactory results for all the chosen validation cases. But a somewhat smaller constant is preferable for the centerline temperature and velocity profiles in the McCaffery simulations. With $C_{LES} = 0.015$ the maximum temperature i over estimated. The slope change in flame height around $Q^* \approx 1$ was not captured by the Vreman and Deardorff turbulence model. The already existing combustion had the same difficulties. LES-EDC with the Smagorinsky model in addition to the existing combustion did also capture the dip for vertical velocity profile in the Sandia plume experiment test 17. But the disadvantage with the Smagorinsky model is the CPU clock times compared with the two other turbulence models.

The errors of the implemented code (referred as FDS-EDC) are in most cases less or the same as for the existing model. The models are also about the same computational expensive. In contrast to the existing model, the implemented code is strongly grid dependent. So before FDS-EDC can be applied in fire analysis the model must be modified to be grid independent. A dynamic constant is not necessary for buoyancy-driven fires in fire engineering application but a more accurate constant is recommended to be established. Temperature and velocity should in further work be validated in a wider range of Q^* for practical fire sizes than in this thesis.

Nomenclature

A	number of collisions	[-]
a	number of nitrogen atoms	[-]
C	model constant	[-]
C	consentration	[mol/m^3]
C_D	Deardorff model constant	[-]
C_{LES}	EDC-LES model constant	[-]
C_S	Smagorinsky model constant	[-]
C_V	Vreman model constant	[-]
c_p	heat capacity	[J/kg·K]
c	speed of light	[m/s]
D	mass diffusivity	[m^2/s]
D	diameter	[m]
D^*	non-dimensional fire diameter	[-]
Da	the turbulent Damköhler number	[-]
E_a	activation energy	[J]
E	thermal radiative energy	[W/m^2]
f	acceleration	[m/s^2]
Fr	Froude number	[-]
g	gravitational constant	[m/s^2]
H	heat	[kJ]
ΔH_c	heat of combustion	[kJ/kg]
h	specific enthalpy	[kJ/kg]
h	convective heat transfer coefficient	[W/m^{2}K]
h	Planck's constant	[Js]
j	diffusiv mass flux	[kg/m^{2}s]
k	number of reactions each second	[s^{-1}]
k	model constant	[-]
k	turbulent kinetic energy	[kJ/kg]
k	thermal conductivity	[W/m·K]
k	Boltzman constant	[J/K]
Ka	the turbulent Karlovitz number	[-]
L	characteristic length scale	[m]
L	length	[m]

L_v	heat required to produce volatiles	[kJ/g]
l	length scale	[m]
M	mass	[kg]
M	molar mass	[g/mol]
Ma	Mack number	[-]
m	mass	[kg]
N	number of species	[-]
Nu	Nusselt number	[-]
p	pressure	[N/m^2]
$\dot{Q}_c$	heat release rate (HRR)	[W]
$\dot{Q}_r$	radiative heat transfer	[W]
Q^*	dimensionless heat release rate	[-]
q	heat	[kJ/kg]
R	universal gas constant	[J/mol·K]
R	reaction rate	[kg/s]
Re	Reynolds number	[-]
R_l	turbulent Reynolds number	[-]
S	strain rate	[s^{-1}]
S	source term	
Sc	Schmidt number	[-]
s	stoichiometric coefficient	[-]
T	temperature	[K]
t	time	[s]
U	internal energy	[J]
u	velocity	[m/s]
W	molecular wight	[kg/kmol]
w	mechanical work	[kJ/kg]
x	direction x	
x	number of carbon atoms	[-]
Y	mass fraction	[kg/kg]
y	direction y	
y	number of hydrogen atoms	[-]
Z	mixture fraction	[kg/kg]
z	direction z	
z	number of oxygen atoms	[-]
z	height	[m]

Greek symbols

α	thermal diffusivity	[m^2/s]
β	model constant	[-]
γ_λ	ratio of fine structure mass between large eddies to the total mass	[kg/kg]
γ^*	ratio of fine structure mass to the total mass	[kg/kg]
Δ	grid cell size	[m]
δ_{ij}	Kronecker-delta ($i = j \rightarrow \delta_{ij} = 1$ and $i \neq j \rightarrow \delta_{ij} = 0$)	[-]
ϵ	dissipation rate of turbulent kinetic energy k	[W/kg]
η	model constant	[-]
λ	wave length	[m]
μ	dynamic molecular viscosity	[kg/m·s]
ν	kinematic molecular viscosity	[m^2/s]
ν	yield	[kg/kg]
ρ	density	[kg/m^3]
σ_k	Schmidt number for turbulent kinetic energy k	[-]
σ_ϵ	Schmidt number for dissipation rate of turbulent kinetic energy ϵ	[-]
σ	Stefan-Boltzman constant	[W/m^{2}K^4]
τ	viscous shear tensor	[N/m^2]
τ	time scale	[s]
φ	varable in transport equation or filtered value	[-]
χ	efficient coefficient	[-]
$\dot{\omega}$	reaction rate	[kg/s]
ω	strain	[s^{-1}]

Superscripts

$'$	fluctuating value
$','' $	characteristic turbulent scale
$''$	per m^2
$\bar{}$	favre average value
$\tilde{}$	filtered value
$\dot{}$	per second
$*$	fine structure
0	surrounding

Subscripts

0	integral length scale
0	centerline
∞	ambient
a	number of nitrogen atoms
d	diffusion
F	fuel
F	flame
f	formation
f	flame
f	fluid
g	gravitational
i	component in x-direction
j	species j
j	component in y-direction
k	component in z-direction
k	species k
k	turbulent kinetic energy
K	Kolmogorov (length scale)
LFT	lower flame temperature
L	laminar
L	loss
n	number
P	products
r	radiation
SGS	subgrid scale
S	soot
s	surface
T	Taylor (length scale)
T	Turbulent
t	turbulent
u	turbulent
v	vapourizing

x	number of carbon atoms
y	number of hydrogen atoms
z	number of oxygen atoms
α	number
ϵ	dissipation rate of turbulent kinetic energy

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Accidental fires and explosions are a severe threat to human life and expensive process equipment in the process industries. To obtain an acceptable risk when handling flammable gas and liquid, knowledge to the physical processes involved in a fire or explosion is required for safety designing. Computational Fluid Dynamics (CFD) is a useful engineering tool to evaluate potential consequences, such as heat loads on process equipment or gas dispersion from accidental leaks, detector and mitigation optimization. Over the last decades, the use of CFD has increased as a result of increased computational power. Increasing computational power is also allowing continuously development of CFD.

Several commercial and non-commercial CFD codes exist for different use. Mainly the codes are divided in which way the turbulence is modeled. Fire Dynamic Simulator (FDS) is a non-commercial Large Eddy Simulation (LES) code developed by NIST (National Institute of Standard and Technology) appropriate for indoor building fire analysis [4]. FDS is limited to simulations of thermal driven flows as buoyancy-driven fires, i.e FDS is unsuitable for jet fire and explosion simulations.

Correct modeling of the combustion process in CFD is crucial to achieve reliable results. Combustion models are based on certain assumptions related to the simulated flame type. The Eddy Dissipation Concept (EDC) proposed by Magnussen and Hjertager, is based on the turbulent mixing to model the reaction rate [1][2][3]. The model assumes that the chemical reaction occurs where reactants and hot products are molecularly mixed. The mixing process is located where the kinetic energy is dissipated into heat. These regions, referred as fine structure, are treated as a perfectly stirred reactor.

EDC in its original form assumes full turbulence cascading in each numerical cell, as the way turbulence is modeled in the Reynolds Average Navier Stokes (RANS). Extension of EDC to LES has been performed by Hu *et al.* and Zhou *et al.*, both with unsatisfactory predictions [7][8]. The reason for the unsatisfactory predictions was that RANS model constants were used. Panjwani *et al.* proposed two approaches to LES-EDC, where the fine structure regions are based on subgrid viscosity instead of turbulent kinetic energy (TKE) and dissipation, because they are usually not solved explicitly in

LES codes [5][6]. A parameter study in their validation showed that the model constant $C_{LES} = 0.25$ was satisfactory for the Smagorinsky turbulence model. However, this was for a jet fire strongly influenced by momentum. The Froude number for the jet flames Panjwani *et al.* studied were in order of 10^4 while fires interested in context of buildings are in order of less than 10^-3.

The objective of this thesis has been to implement LES-EDC in FDS and validate the implemented code (hereafter referred as FDS-EDC) to the application area of FDS. More precisely the objective has been to:

- establish a model constant for thermal buoyancy-driven fires and evaluate whether a static constant is sufficient with respect to different fire sizes or a dynamic constant is necessary
- establish a model constant for all supported turbulence models in FDS6; Vreman, Smagorinsky and Deardorff
- evaluate if LES-EDC can replace the already existing combustion model in FDS (hereafter referred as FDS6)

The simulations were limited to two parameter mixture fraction with a infinitely fast single-step reactions for non-premixed flames, i.e detailed kinetics are outside the scope of this thesis.

Chapter 2

Theory

2.1 Computational Fluid Dynamics

Computational fluid dynamics (CFD) is a powerful engineering tool, which is used to analyze fluid flow problems. CFD is used in a wide range of engineering disciplines, from designing airplanes, studying pipeline flows to fire engineering applications. Ever-increasing computational power leads to continuously development of CFD, which over time allows implementing of more complex algorithms. Because of inaccuracy in numerical techniques and limited precision in representation of numbers in a computer, CFD should always be considered as a supplement for experimental investigation and not a substitute [9]. In general, CFD is deterministic and is the reason why results are totally dependent on input values as boundary conditions and specified transient behavior in the computational domain. This leads to correct assumptions has to be taken into account. Therefore, CFD requires expert knowledge in physics, chemistry, computer science and numerical methods [10].

2.1.1 Governing Equations

In CFD, the finite volume method is applied by dividing a computational domain in to several sub volumes, a grid, where partial differential equations are solved in center of each sub volume or grid cell. The partial differential equations (PDE) are based on conservation of mass, momentum, energy and other static variables, φ. A general flow equation is given by

$$\underbrace{\frac{\partial}{\partial t}(\rho\varphi)}_{\text{accumulation}} + \underbrace{\frac{\partial}{\partial x_j}(\rho\varphi u_j)}_{\text{convective transport}} = \underbrace{\frac{\partial}{\partial x_j}(-j_{\varphi,j})}_{\text{diffusive transport}} + \underbrace{S_\varphi}_{\text{source term}} \tag{2.1}$$

The first term is accumulation of φ, the second is convective transport of φ, the third term is diffusive transport of φ and the last is the source term. Since an analytical solution on these equations does not exist, the equations are solved numerically. Taylor expansions series is the most common technique to solve these partial differential equations [9]. The governing equations are:

Conservation of mass (Continuity Equation)

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_j) = 0 \tag{2.2}$$

Conservation of momentum (Newton's Second Law)

$$\frac{\partial}{\partial t}(\rho u_j) + \frac{\partial}{\partial x_j}(\rho u_i u_j) = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho f_i \tag{2.3}$$

Conservation energy (First Law of Thermodynamics)

$$\frac{\partial}{\partial t}(\rho h) + \frac{\partial}{\partial x_j}(\rho u_i h) = \frac{\partial}{\partial x_j}\left(\rho\alpha\frac{\partial h}{\partial x_j}\right) + \dot{Q}_r + \rho\dot{\omega}Y_f\Delta H_c \tag{2.4}$$

These five partial differential equation consist six solution vectors: density ρ, temperature T, pressure p, three velocity components u_i, u_j and u_k. Note that it is three equations for conservation of momentum, one in each direction. The enthalpy is given as a function of temperature; $h = \int_{T_0}^{T} c_p T dT$. Pressure is calculated from the equation of state

$$p = \frac{\rho R T}{M} \tag{2.5}$$

Additional sub models are needed for closing the source terms, e.g radiation, Reynolds Stresses and combustion.

2.1.2 Reacting Flows

For reacting flows or multi-species flows, mass fractions for each species, k, is conserved

$$\frac{\partial}{\partial t}(\rho Y_k) + \frac{\partial}{\partial x_j}(\rho u_i Y_k) = \frac{\partial}{\partial x_j}\left(\rho D\frac{\partial Y_k}{\partial x_j}\right) + R_k \tag{2.6}$$

From the relation $\sum_k Y_k = 1$ the last species can be calculated, hence $N-1$ partial differential equations are necessary for a total number N species.

PDE are expensive to compute. So to reduce the number of PDE when computing reacting flow, the mixture fraction can be computed instead of mass fractions:

$$\frac{\partial}{\partial t}(\rho Z_k) + \frac{\partial}{\partial x_j}(\rho u_i Z_k) = \frac{\partial}{\partial x_j}\left(\rho D\frac{\partial Z_k}{\partial x_j}\right) + R_k \tag{2.7}$$

Considering two streams mixing together, Z kg with property φ_1 in stream 1 and $(1-Z)$ kg with property φ_2 in stream 2 giving 1 kg mixture $\varphi_{mix} = Z\varphi_1 + (1-Z)\varphi_2$. The mixture fraction is then

$$Z = \frac{\varphi_{mix} - \varphi_2}{\varphi_1 - \varphi_2} \tag{2.8}$$

Assuming no sink or source, this expression can be used for all kinds of scalars [11]. In combustion modeling, stream 1 could be fuel and stream 2 air with property mass fractions $(Y_k)_{mix} = Z(Y_k)_1 + (Z-1)(Y_k)_2$. Assuming stoichiometric infinitely fast chemistry of

$$1 \text{ kg fuel} + s \text{ kg air} \rightarrow (1+s) \text{ products} \tag{2.9}$$

"mixed is burned", i.e oxygen and fuel cannot coexist, the scalar $Y_F - \frac{1}{s}Y_{O_2}$ is conserved for a comlplete reaction. Y_F and Y_{O_2} are mass fractions for fuel and oxygen respectively, and s is the stoichiometric coefficient. At stoichiometric condition, the mixture fraction is

$$Z_{stoich} = \frac{(Y_F - \frac{1}{s}Y_{O_2})_{stoic} - (Y_F - \frac{1}{s}Y_{O_2})_2}{(Y_F - \frac{1}{s}Y_{O_2})_1 - (Y_F - \frac{1}{s}Y_{O_2})_2} = \frac{1}{s+1} \tag{2.10}$$

Then $Y_{products} = 1$ and there is no fuel or oxygen left. In the fuel-rich region $(Z > Z_{stoich})$ no oxygen is left, and in the fuel-lean region $(Z < Z_{stoich})$ no fuel is left. Now, all mass fractions can simply be calculated.

2.1.3 Turbulence and Turbulence Modeling

To give a precise definition of turbulence is very difficult, but in fluid dynamics may turbulent flows be characterized as irregular motion of fluid where velocity is rapidly fluctuating. The underlying physical mechanism in turbulence is complex and is yet not fully understood. Turbulence remains as one of the most important unsolved physical problem, because of the fact that almost all natural flows are turbulent [12]. To evaluate whether a flow is turbulent or laminar, the dimensionless Reynolds number is calculated. The Reynolds number is the ratio of inertial forces to viscous forces. In turbulent flows inertial forces are dominating, hence the Reynolds number is large.

In turbulent flows, rotational flow structures called eddies are formed, each with different characteristic length and velocity scale ($'$,$''$,$'''$,...,*). Eddies are formed as a consequence of shear stress (friction) generated between fluid sheets of different velocities. Turbulent flows are always dissipative [12]. Eddies are dissipating through vortex stretching caused by viscous shear stress in the flow and the kinetic energy is converted to internal energy (see Figure 2.1). Mechanic energy is transferred from the main stream to large eddies and further down to smaller eddies [13]. The eddy dissipation continues down to a level where the diffusion time across the diameter equals the time for an eddy to rotate 1/2 revolution [14]. At this level, the *length scale* of an eddy equals *Kolmogorov length scale* and is where molecular mixing dominating (see Figure 2.2). This breakdown of eddies is called the energy cascade model and is seen in Figure 2.8.

The major part of the kinetic energy is in the large eddies [14]. In turbulent flows for high turbulence Reynolds number large eddies are much larger than small eddies. According to Kolmogorov the small eddies are then not influenced by large eddies and the main stream. A typical distribution of *turbulent kinetic energy* is given in Figure 2.2.

Integral length scale, l_0, is the largest length scale and corresponds to geometrical dimensions [14]. Large eddies has characteristic scales as the mean flow and are therefore

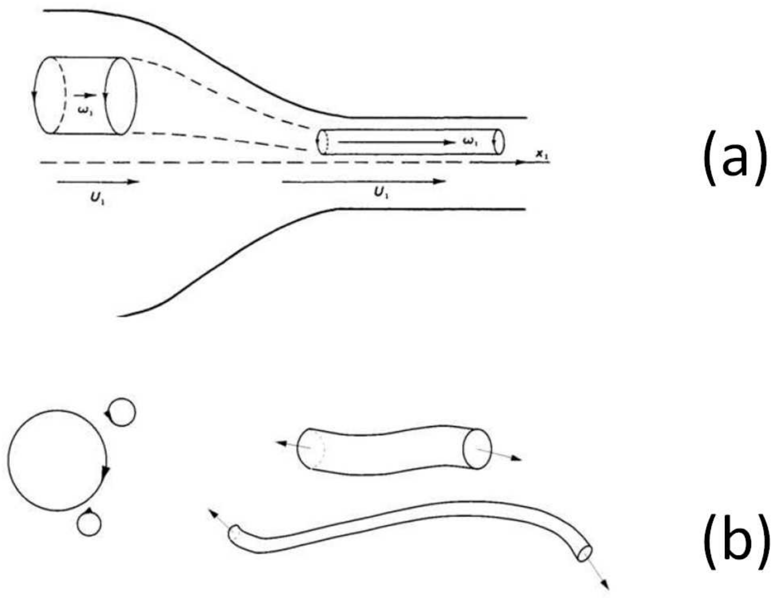


Figure 2.1: (a) Vortex streching in wind tunnel [12]. Angular momentum is conserved. (b) Principle of vortex stretching [13].

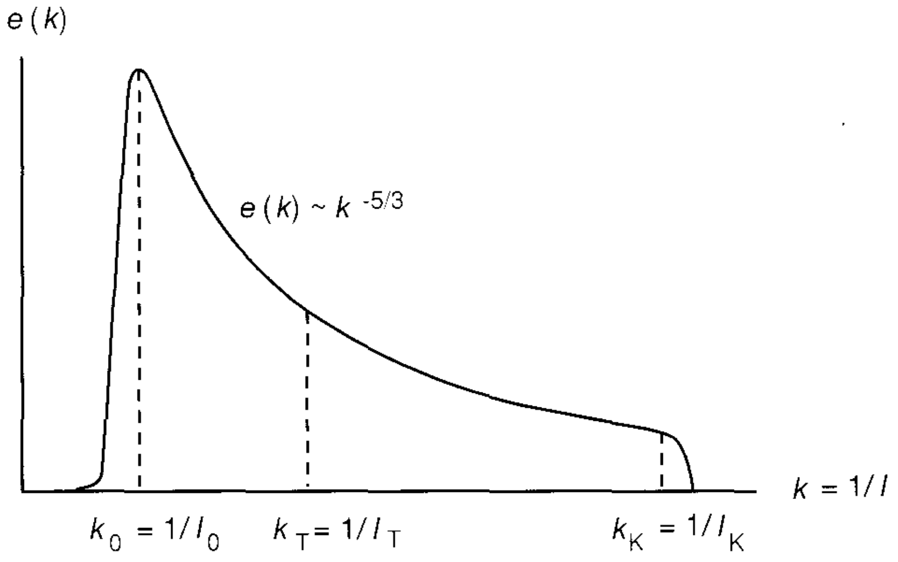


Figure 2.2: Energy cascade: Distribution of *turbulent kinetic energy* at different length scales [14]. Integral length scale, l_0, Taylor length scale, l_T, and Kolmogorov length scale, l_K.

dominated by inertia effects [15]. I.e large eddies are inviscid in contrast to small eddies which are influenced by viscosity. For this reason, high viscosity flows are associated with few large eddies while low viscosity flows are associated with less larger eddies but a larger amount of smaller eddies. The largest length scale where viscosity affects the dynamics of turbulent eddies is called Taylor length scale, l_T.

Depending on how the turbulence is treated, CFD models are divided in

- Reynolds Average Navier-Stokes (RANS)
- Large Eddy Simulation (LES)
- Direct Numerical Simulation (DNS)

In RANS, all turbulent structures are modeled in a sub grid scale model (SGS) , in contrast to DNS where fluctuations are captured on the grid points and calculated directly by the governing equations. Between DNS and RANS, LES are capturing large eddies on the grid points while small eddies are modeled.

In the momentum equation (eq. (2.3)), the viscous stress tensor is divided in $\tau_{eff} = \tau_{mol} + \tau_{turb}$, where τ_{turb} can be regarded as an extra stress (or viscosity) caused by increased transport in turbulent flows. This extra stress is closed by a turbulence model. According to Boussinesq hypothesis, Reynolds stresses ($-\bar{\rho}\widetilde{u_i'u_j'}$, $-\bar{\rho}\widetilde{u_j'h'}$ and $-\bar{\rho}\widetilde{u_j'Y_k'}$) are proportional to the mean rate of strain:

$$-\bar{\rho}\widetilde{u_i'u_j'} + \frac{2}{3}\bar{\rho}\tilde{k}\delta_{ij} = 2\mu_t\tilde{S}_{ij} - \frac{2}{3}\mu_t\frac{\partial \tilde{u}_k}{\partial x_k}\delta_{ij} \tag{2.11}$$

where μ_t is the *turbulent viscosity*, δ_{ij} is the Kronecker delta ($i = j \to \delta_{ij} = 1$ and $i \neq j \to \delta_{ij} = 0$) and turbulent kinetic energy $\tilde{k} = \frac{1}{2}\widetilde{u_i'u_j'}$. Bars over the variables represents favre filtered quantities (for LES) or average quantities (for RANS). The strain rate tensor is expressed as

$$\tilde{S}_{ij} = \frac{1}{2}\left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i}\right) \tag{2.12}$$

Reynold's Average Navier-Stokes (RANS)

RANS is applied when only average values are interesting in fluid flow analysis. RANS is the least computational expensive model and is used in most commercial CFD codes. In RANS, the solution vector is split in a fluctuating term and an average term, e.g for velocity $u = \bar{u} + u'$. The most common model to close RANS equations is k-ϵ two-equation model, where two additional partial differential equations for transport of *turbulent kinetic energy*, k, and *dissipation rate* of kinetic energy, ϵ, are solved:

$$\frac{\partial(\bar{\rho}k)}{\partial t} + \frac{\partial(\bar{\rho}\bar{u}_j k)}{\partial t} = \frac{\partial}{\partial x}\left[\frac{\mu_T}{\sigma_k}\frac{\partial k}{\partial x_j}\right] - \bar{\rho}\widetilde{u_i'u_j'}\frac{\partial \tilde{u}_i}{\partial x_j} - \bar{\rho}\epsilon \tag{2.13}$$

$$\frac{\partial(\bar{\rho}\epsilon)}{\partial t} + \frac{(\partial\bar{\rho}\bar{u}_j\epsilon)}{\partial t} = \frac{\partial}{\partial x}\left[\frac{\mu_T}{\sigma_\epsilon}\frac{\partial k}{\partial x_j}\right] - C_{\epsilon 1}\frac{\epsilon}{k}\left(\bar{\rho}\widetilde{u_i'u_j'}\frac{\partial \tilde{u}_i}{\partial x_j}\right) - C_{\epsilon 2}\bar{\rho}\frac{\epsilon^2}{k} \tag{2.14}$$

For the two-equation model the turbulent viscosity is

$$\mu_t = \bar{\rho} C_\mu \frac{k^2}{\epsilon}, \tag{2.15}$$

where $k = \frac{1}{2}\widetilde{u_i' u_i'}$ and $\epsilon = \frac{\mu_T}{\bar{\rho}} \widetilde{\left(\frac{\partial u_i'}{\partial x_j}\right)\left(\frac{\partial u_i'}{\partial x_j}\right)}$. The constants for eq.(2.13), (2.14) and (2.15) have been established by data fittings from experiments of a broad range of turbulent flows; $C_\mu = 0.09$, $\sigma_k = 1.0$, $\sigma_\epsilon = 1.3$, $C_{\epsilon 1} = 1.44$, $C_{\epsilon 2} = 1.92$ [16]. However, the one-equation model only consist one additional partial differential equation. This equation is for conservation of *turbulent kinetic energy.* In the zero-equation model, the turbulent viscosity is calculated directly from an empirical correlation.

Other turbulence models for RANS equations may be k-ω, Menter model, and the more sophisticated Reynolds Stress Model which involves calculation of individual Reynolds stresses and gives seven additional partial differential equations.

Large Eddy Simulation (LES)

The computational time of Large Eddy Simulation is less than DNS and more than RANS. LES compute explicitly the largest structures in the flow by filtered Navier-Stokes equation, while small structures, typically smaller than the grid cell size, are modeled. The grid cell size must be in order that a sufficient amount of the turbulent energy is solved on the grid points (see Section 3.4). In 1D, large structure (low wave numbers) are filtered by

$$\tilde{\varphi}(x,t) = \int_{x-\frac{\Delta}{2}}^{x+\frac{\Delta}{2}} \varphi(r,t)dr \tag{2.16}$$

where $\tilde{\varphi}$ is the filtered quantity and Δ is the local cell size.

The most popular approach for the unresolved fluxes in LES is the Smagorinsky model:

$$\mu_t = \rho C_S^2 \Delta^{4/3} l_t^{2/3} |\bar{S}| \tag{2.17}$$

where $\Delta = (\delta x \delta y \delta z)^{\frac{1}{3}}$ and the strain rate, S, is given in eq. (2.12). The empirical constant, C_S, is usually 0.1 - 0.2. Eq. (2.17) is simplified assuming that the turbulence integral length scale is in the same order as the grid cell size, $l_t \approx \Delta$:

$$\mu_t = \rho (C_s \Delta)^2 |\bar{S}| \tag{2.18}$$

Germano et al. attempted to formulate a more universal approach, and proposed a dynamic Smagorinsky constant [17]. A detailed description of the dynamic constant is found in various litterature (e.g Theoretical and Numerical Combustion, 2005 [18].

Another model turbulence model is the Deardorff eddy viscosity model. The eddy viscosity is expressed as

$$\mu_T = \rho C_D \Delta \sqrt{k_{SGS}}, \tag{2.19}$$

where $C_D = 0.1$ (in FDS v.6) and k_{SGS} is the sub grid scale kinetic energy.

Recently, Vreman proposed a simple turbulence model that similar to the Smagorinsky model, only needs the first-order derivatives of the velocity and the local filter width to compute the viscosity [19]:

$$\mu_T = \rho C_V \sqrt{\frac{B_\beta}{\alpha_{i,j}\alpha_{i,j}}} \tag{2.20}$$

$C_V \approx 2.5 C_S$. For homogenious isotropic turbulence, the Smagorinsky constans equals 0.17 and gives $C_V = 0.07$ (default in FDS v.6) [19]. α is a 3×3 matrix with derivatives of the filtered velocity:

$$\alpha = \begin{bmatrix} \frac{\partial \tilde{u}}{\partial x} & \frac{\partial \tilde{u}}{\partial y} & \frac{\partial \tilde{u}}{\partial z} \\ \frac{\partial \tilde{v}}{\partial x} & \frac{\partial \tilde{v}}{\partial y} & \frac{\partial \tilde{v}}{\partial z} \\ \frac{\partial \tilde{w}}{\partial x} & \frac{\partial \tilde{w}}{\partial y} & \frac{\partial \tilde{w}}{\partial z} \end{bmatrix} \tag{2.21}$$

By definition, μ_T is set to zero if $\alpha_{i,j}\alpha_{i,j} = 0$. Further,

$$B_\beta = \beta_{11}\beta_{22} - \beta_{12}^2 + \beta_{11}\beta_{33} - \beta_{13}^2 + \beta_{22}\beta_{33} - \beta_{23}^2 \tag{2.22}$$

where $\beta_{i,j} = \Delta_m^2 \alpha_{m,i}\alpha_{m,j}$.

Direct Numerical Simulation (DNS)

DNS is the most computational expensive approach where the transport equations are solved numerically without any turbulence model. To capture all the spatial scales of turbulence on the numerical grid, from *Kolmogorov length scale*, l_k (smallest dissipative scales), to *integral length scale*, l_0, a sufficient number of grid points are needed. According to J. Warnatz et al. [14] about 1000 grid points in each direction are necessary to resolve the smallest turbulent structures for a typical flow with turbulence Reynolds number $R_l = 500$ and ratio $l_0/l_k \approx 100$. This equals for a three-dimensional simulation a total of 10^9 grid points. For RANS and LES a typical range in order of 10^5-10^6 grid points are sufficient. Therefore, DNS is limited to small-scale laminar bench scale flames in the matter of fire research.

2.2 Combustion

Combustion is an exothermic process where fuel and oxidant are reacting chemically involving rather complex physics and chemistry. The combustion can be seen as light, either as a flame or a glow, caused by the heat released in the reactions. Simplified, fuel and an oxidant are reacting forming carbon dioxide, carbon monoxide, soot and water vapor for combustion of hydrocarbons:

$$fuel + oxidant \rightarrow H_2O + CO + CO_2 + soot \tag{2.23}$$

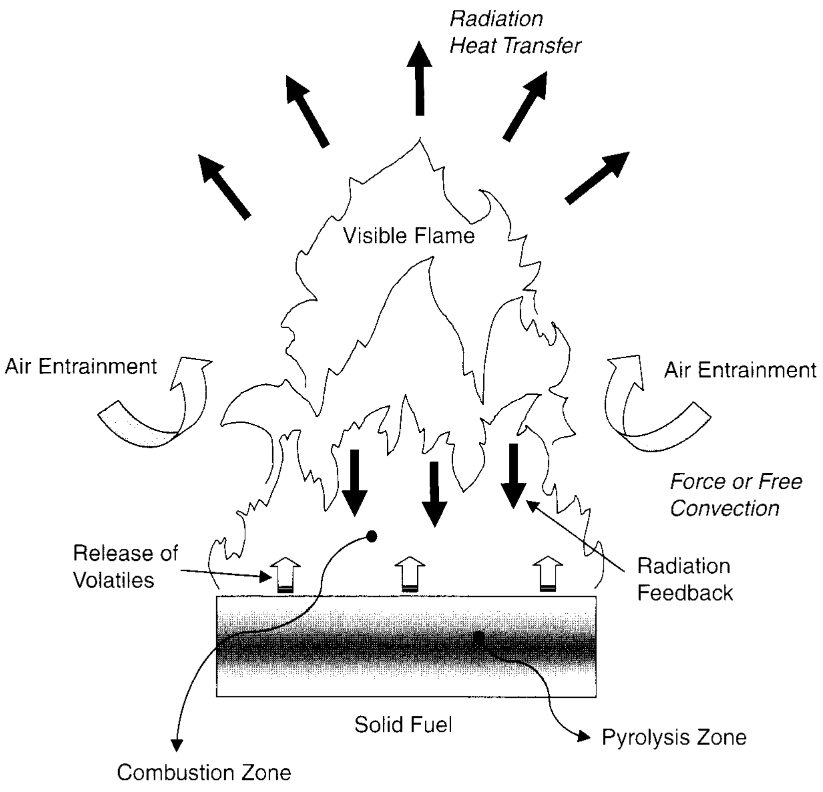


Figure 2.3: Physical processes involved in a fire [15].

In natural fires, oxygen in the air is the oxidant in the chemical reactions. The fuel can be solid, gas or liquid usually carbon-based. However, the flame is a gas phase phenomenon [20]. This means that in a combustion process involving a solid or liquid, a conversion to gaseous form is necessary. In a liquid fire the radiation from the flame is evaporating the liquid, but for most solids, a chemical decomposition or *pyrolysis* takes place instead of evaporation. *Pyrolysis* yields products light enough to volatilize from the surface before entering the combustion zone. A general mass burning rate be expressed as

$$\dot{m}'' = \frac{\dot{Q}''_F - \dot{Q}''_L}{L_v} \tag{2.24}$$

where L_v is the heat required for evaporation or *pyrolysis*, $\dot{Q}''_F$ the heat supplied by the flame to the fire area and $\dot{Q}''_L$ heat loss from the fuel surface. A visual description of physical processes in a fire is shown in Figure 2.3.

2.2.1 Chemical Kinetics

A combustion process consist of several chemical reactions, called *elementary reaction*. Even a simple combustion of methane in air consists 400 reactions [21]. The reactions of interest, *radical chain reactions*, are divided in *chain propagation*, *chain branching* and *chain termination*. Warnatz *et al.* considered a hydrogen combustion where the most important reactions are given in Table 2.1 [14]. An overall stoichiometric reaction for combustion of hydrogen in oxygen is:

$$2H_2 + O_2 \rightarrow 2H_2O \tag{2.25}$$

Table 2.1: Chemical reactions with respect to hydrogen ignition (adapted from Warnatz *et al.*, 2006 [14])

Reaction number		Chemical reaction	Reaction Mechanism
0	$H_2 + O_2$	$= 2OH\bullet$	*chain initiation*
1	$OH \bullet + H_2$	$= H_2O + H\bullet$	*chain propagation*
2	$H \bullet + O_2$	$= OH \bullet + O\bullet$	*chain branching*
3	$O \bullet + H_2$	$= OH \bullet + H\bullet$	*chain branching*
4	$H\bullet$	$= 1/2H_2$	*chain termination*
5	$H \bullet + O_2 + M$	$= HO_2 + M$	*chain termination*

The dots in the reactions represents that the species are a free radical. Free radicals are highly reactive because of its charge and are essential to obtain the chemical reactions in a combustion process. In the first reaction, *chain initiation*; a stable species are forming one reacting species. The *chain initiation* leads to *chain propagation* (reaction 2 and 3) where a reactive species react with stable species forming another reactive species. In *chain branching steps* two reactive species are formed from a stable species and a reactive species. The last steps (reaction 4 and 5), *chain termination*; no reactive species are formed.

2.2.2 Heat Release Rate (HRR) and Reaction Rate

The heat release rate (HRR) is the most important factor to characterize a fire. Mathematically HRR can be expressed as

$$\dot{Q}_c = \chi \dot{m} \Delta H_c \tag{2.26}$$

where χ is an efficiency coefficient which takes into account that the combustion is incomplete, and heat is lost to surroundings e.g. through radiation and heat transfer to the air entrained (see figure 2.3). $\dot{m}$ is the fuel consumption rate. The reaction rate is depended on a characteristic time scale limiting the reaction. A simple assumption would be finite chemical reaction rate, hence the chemical time scale limits the reaction rate. Collisions between molecules is the underlying phenomena in chemical reactions, i.e the reaction rate is a function of temperature, pressure and concentration. The reaction rate may be given as

$$R_k = -k\rho^2 C_{fuel} C_{oxygen} \tag{2.27}$$

where C_{fuel} and C_{oxygen} are concentrations of fuel and oxygen respectively. k is the rate coefficient are given by *Arrhenius law* for temperature depended reactions:

$$k = Ae^{-E_a/RT} \tag{2.28}$$

A is the total number of molecular collisions and $e^{-E_a/RT}$ the fraction of collisions that leads to reactions. Figure 2.4 shows that E_a, the *activation energy*, is the energy barrier to start the reaction. R is the *universal gas constant* and T absolute temperature in Kelvin. A more sophisticated method assumes infinitely fast chemistry and would (also) involve other limiting time scale e.g turbulent time scale, diffusive timescale or acceleration time scale. See section 2.5 and 3.1.

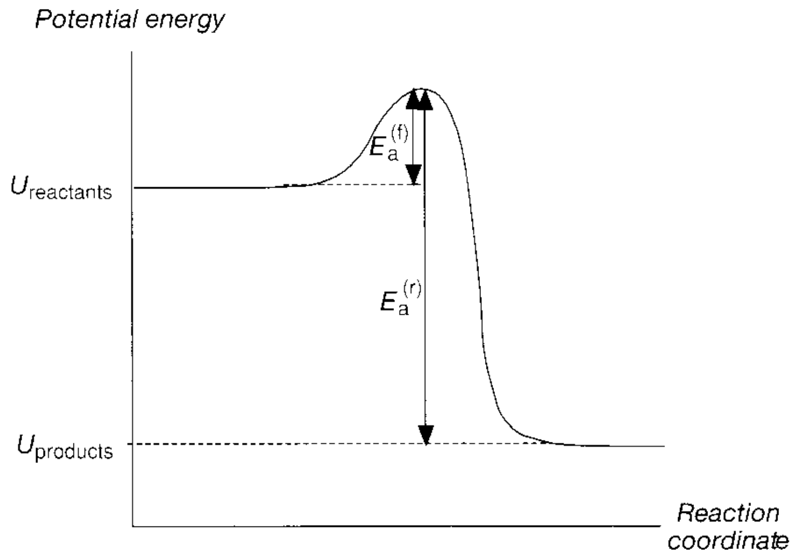


Figure 2.4: A graphical description of energy in an exothermic reaction [14]. $E_a^{(f)}$ is the energy barrier to start the reaction. Considering a combustion $\Delta H_c \approx U_{reactants} - U_{products}$.

ΔH_c is the *heat of combustion*, the energy released during combustion per unit mass or mole. Heat of combustion equals the difference in energy of species before the combustion and after the combustion. Assuming adiabatic conditions (no heat loss to surroundings), the heat of combustion can be determined experimentally in calorimeter bomb at constant volume, where temperature rise is measured.The change in enthalpy according to the first law of thermo dynamics is then

$$\Delta H = \Delta U + p\Delta V \tag{2.29}$$

From the ideal gas law (eq. (2.5)) the work done ($p\Delta V$) can be calculated. But since the work done is small compared to the increase of internal energy (ΔU), it may be neglected, leaving $\Delta H_c \approx U_{products} - U_{reactants}$, as shown in Figure 2.4.

When a compound is formed under standard state (1 atm. pressure and 298 K), the change of enthalpy is by definition, the same as heat of formation (ΔH_f) [20]. Considering stoichiometric combustion of methane, the balanced reaction is

$$CH_4 + 2O_2 \rightarrow 2H_2O + CO_2 \tag{2.30}$$

The heat of combustion may be calculated by definition of heat of formation

$$\Delta H_c(CH_4) = (2\Delta H_f(H_2O) + \Delta H_f(CO_2)) - (2\Delta H_f(O_2) + \Delta H_f(CH_4)) \tag{2.31}$$

ΔH_f for reactants and products are found in various literature.

2.3 Flame Characteristics and Fire Plumes

Flames are categorized in four different types given in Table 2.2. When fuel and the oxidant are mixed and burned simultaneously, the flame is *non-premixed.* If the fuel and oxidant are mixed before the combustion it is called a *premixed* flame. Non-premixed and premixed flames are either turbulent or laminar. As flows in general, most flames are also turbulent. Rapid mixing of reactants in the combustion zone for turbulent flames increases the reaction rate and is characterized by an irregular flame sheet.

Table 2.2: Flame types (adapted from Warnatz *et al.*, 2006 [14]).

Fuel/Oxidizer Mixing	Fluid Motion	Example
premixed	laminar	flat flame Bunsen burner
	turbulent	gasoline engine and gas turbines
nonpremixed	laminar	wood fire and candles
	turbulent	aircraft turbine, diesel engine and coal combustion

Froude number is a way to characterize a fire and is expressed as

$$Fr = \frac{U^2}{gD} \tag{2.32}$$

and is the ratio of momentum forces and buoyancy forces. U is the gas velocity, D fire diameter or a characteristic length and g gravitational constant. The velocity is estimated by

$$U = \frac{\dot{Q}_c}{\Delta H_c \rho(\pi D^2/4)} \tag{2.33}$$

where $(\pi D^2/4)$ is the diameter, $\dot{Q}_c$ the HRR, ΔH_c the heat of combustion and D the fire diameter. Dimensionless HRR is a function of the Froude number and is in fire dynamics often replaced by the Froude number. The dimensionless heat release rate was first introduced by Zukoski and others and is expressed as

$$Q_c^* = \frac{\dot{Q}_c}{\rho_\infty c_p T_\infty \sqrt{gD} D^2} \tag{2.34}$$

where ρ is the density, c_p the heat capacity and T the temperature. The indexes, ∞, represent the ambient value.

Flames dominated by high momentum (high Froude numbers or $Q_c^* > 10^5$, region V in Figure 2.6) fuel release and in practice highly turbulent are called jet flames. Such flames can occur from accidental leaks in pressurized pipelines or vessels at process plants, where hydrocarbons are processed.

In natural fires, buoyancy caused by density difference between rising hot gases in the fire plume and ambient air is the dominant driving force. The buoyancy force, in contrast to the momentum from the fuel flow in jet flames, is resisted by viscous drag working in opposite direction, leads to shear stress in the flame sheet. As a consequence eddies are formed if the buoyancy force is large enough. In this process air is *entrained* in the fire plume and mixes fuel with the air. The flame itself is not influenced by the momentum from the fuel release as for jet flames. Natural fires are associated with relatively low velocity since the radiation from the flame is controlling the fuel release (evaporation or pyrolysis) [22]. As the fire diameter increase the radiation also increases, generally resulting in a larger and more turbulent flame.

2.3.1 Borghi Diagram

The Borghi diagram is a diagram for characterizing of flame regimes, seen in Figure 2.5. u'/u_L versus l_0/l_L is plotted with log-log axis where u' is the velocity fluctuation, u_L the laminar burning velocity, l_0 the integral length scale and l_L the laminar flame thickness.

The diagram is divided by three diagonal lines which represents where the dimensionless numbers Da, Ka and Re_T are unity. $Re_l = Re_T^2$ where the turbulent Reynolds number is

$$Re_l = \frac{u' l_0}{\nu} \tag{2.35}$$

ν is the kinematic viscosity $\nu = \mu/\rho$. *The turbulent Damköhler number*, Da, is large for a fast reaction and small for slow reaction. The number denotes the ratio between macroscopic time scale and the chemical times scale given by

$$Da = \frac{t_0}{t_L} = \frac{l_0 u_L}{l_L u'} \tag{2.36}$$

The turbulent Karlovitz number is expressed as

$$Ka = \frac{t_L}{t_K} \tag{2.37}$$

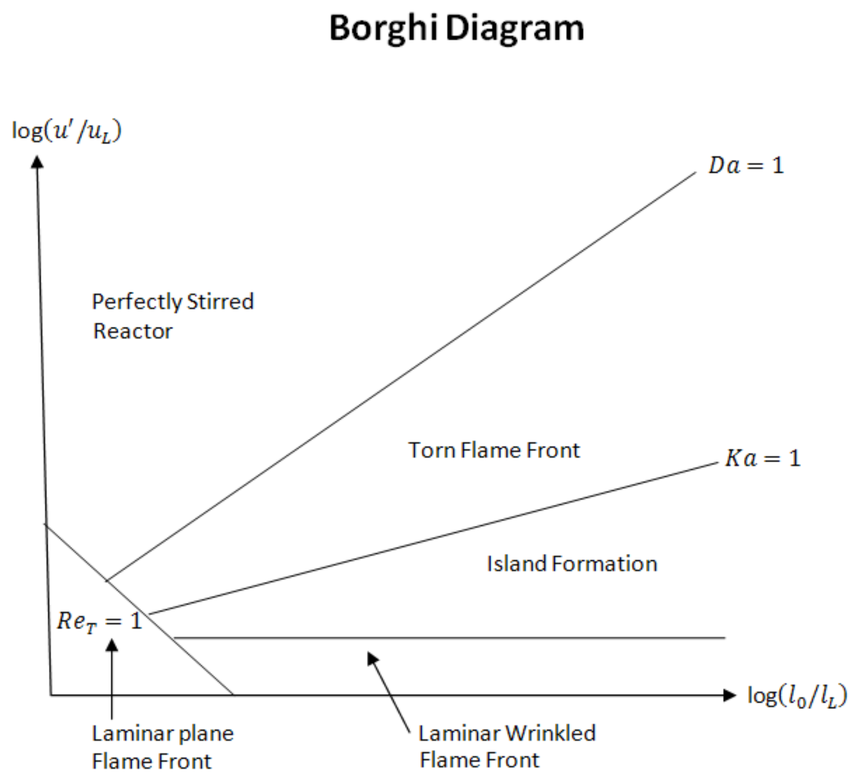


Figure 2.5: Borghi diagram.

where $t_l = l_L/v_L$ and $t_K = \sqrt{\nu/\epsilon}$. t_l and t_K is the time scales of laminar flame and Kolomogorov, respectively.

In the laminar regime $Re_T > 1$, the flame sheet has a thin and a flat reaction zone. Moving in positive x-direction in the Borghi diagram, the turbulence is increasing forming a wrinkled flame front. In the turbulent regime where $Ka < 1$ island of burning fuel in the and air in the flame sheet is observed. Between $Da = 1$ and $Ka = 1$ the chemical time is larger than change of fluid motion [14]. Still, the chemical time is not so large compared to the fluid motion that island formations occur. Above $Da = 1$ the turbulence is so intense compared to the chemical time, allowing a perfect mix fuel and oxidant before the reaction occurs. This regime is referred as *perfectly stirred reactor* as in Section 2.5.3 in the EDC combustion model.

2.3.2 Flame Height

Flame height is an important parameter in fire safety engineering, for example to determine the distance where the radiant heat is sufficient to ignite combustible items. The correlation by Heskestad for buoyancy flames is given by [23]

$$\frac{L_f}{D} = 3.7(\dot{Q}_c{}^*)^{2/5} - 1.02 \tag{2.38}$$

where L_f is the flame height and D is fire area diameter. The dimensionless heat release rate is given in eq. (2.34). The dimensionless flame length, l_f/D, as a function of the dimensionless heat release rate is seen in Figure 2.6. Fire types are divided in five regions. Turbulent buoyancy driven diffusion flames are in regions I and II while jet flame is in region V. For natural fires (i.e not jet fires), Q^* is less than 10 in most situations [10].

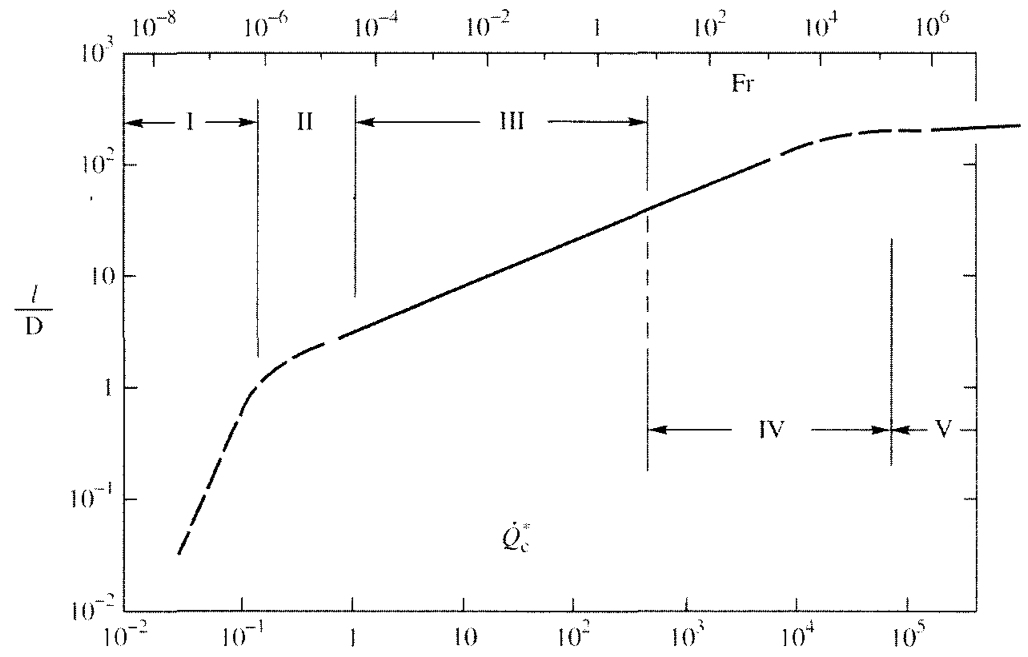


Figure 2.6: The figure is showing the dimensionless flame length, l_f/D, as a function of the dimensionless heat release rate and the Froude number [20].

2.3.3 Centerline Flame Temperature and Velocity

In 1979, McCaffery measured centerline temperature and velocity profiles above 0.3 m square burner [24]. The fuel was methane corresponding to five HRR ranging from 14 kW - 57 kW. McCaffery divided the fire plume in three regimes (in Figure 2.7):

- Persistent flame: accelerating flow
- Intermittent flame: nearly constant flow velocity
- Buoyant plume: decreasing velocity and temperature with respect to height

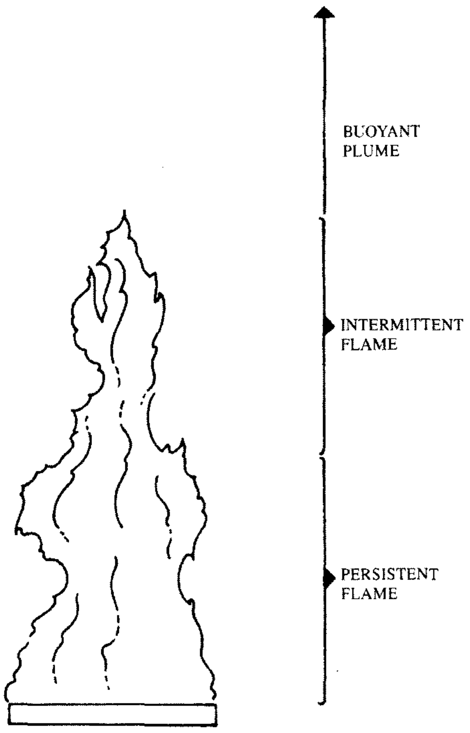


Figure 2.7: McCaffery's fire plumes regimes [20].

From the experiments, McCaffery proposed empirical correlation to centerline velocity, u_0,

$$\frac{u_0}{\dot{Q_c}^{1/5}} = k \left(\frac{z}{\dot{Q_c}^{2/5}} \right)^{\eta} \tag{2.39}$$

and centerline temperature, T_0,

$$\frac{2g\Delta T_0}{T_0} = \left(\frac{k}{C} \right)^2 \left(\frac{z}{\dot{Q_c}^{2/5}} \right)^{2\eta - 1} \tag{2.40}$$

where Q_c is HRR, z is the height above the fire, ΔT_0 is the temperature difference between the centerline temperature and ambient temperature. k, η and C are empirical constants given in Table 2.3.

2.4 Heat Transfer

Heat can be transferred in three different ways: *conduction*, *convection* and *radiation* - all involved in a fire.

2.4.1 Conduction

Heat transfer through a solid or non-moving fluid, due to a temperature gradient, is called conduction. The physical nature of heat is vibrations of molecules, and spreads from regions of higher temperatures to regions of lower temperature, by collisions and diffusion of molecules. According to Fourrier's law of conduction, the steady state one-dimensional heat rate transfer through an area is proportional to the negative temperature gradient

$$\dot{q}''_x = -k\frac{dT}{dx} \tag{2.41}$$

The thermal conductivity, k, is a measure of how well a material can transfer heat. In transient phenomenas such as fires, the non-steady state conduction is expressed as

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) \tag{2.42}$$

where c_p is the heat capacity. Eq. (2.41) and (2.42) assumes constant thermal conductivity, heat capacity and density that in reality is a function of temperature.

2.4.2 Convection

Convection is heat exchange between a moving liquid or gas and a solid and is given by Newton's law of cooling

$$\dot{q}'' = h(T_s - T_\infty) \tag{2.43}$$

where h is the convective heat transfer, T_s the temperature at the surface and T_∞ the surrounding temperature. From the ratio of convective heat transfer coefficient to conductive heat transfer, also known as the dimensionless Nusselt number, the convective heat transfer is determined

$$Nu = \frac{hL}{k_f} \tag{2.44}$$

L is the characteristic length and k_f conductivity of the fluid. The Nusselt number is depended on the fluid property, the thickness of boundary layer created by the shear stress and the flow property. Hence convection involving buoyancy-driven flows as a consequence of temperature gradients are referred as natural convection is distinguished from forced convection involving external forces.

Table 2.3: Empirical constants in McCaffery's centerline temperature and velocity profile

Region	k	η	$z/\dot{Q}^{2/5}$	C
		[-]	[m/kW$^{2/5}$]	[-]
Persistent flame	6.8 m$^{1/2}$/s	1/2	<0.08	0.9
Intermittent flame	1.9 m/kW$^{1/5}$·s	0	0.08-0.2	0.9
Buoyant plume	1.1 m$^{4/3}$/kW$^{1/3}$·s	-1/3	>0.2	0.9

2.4.3 Radiation

Heat transferred by electromagnetic waves is called radiation. From Planck's law, energy emitted by a black body per unit area at a given temperature, T, within a narrow band of wavelengths, λ, is

$$E_{b,\lambda} = \frac{2\pi c^2 h \lambda^{-5}}{\exp(ch/\lambda kT) - 1} \tag{2.45}$$

where c is the speed of light, h the Planck's constant and k the Boltzman constant. Integrating eq. (2.45) over the whole spectrum of wavelengths gives

$$E_b = \int_0^\infty E_{b,\lambda} d\lambda = \frac{2\pi^5 k^4 T^4}{15 c^2 h^3} \tag{2.46}$$

The emissivity, ϵ, is not unity for surfaces other than for black bodies and is defined as

$$\epsilon = \frac{E_\lambda}{E_{b,\lambda}} \tag{2.47}$$

Rewriting eq. (2.46) gives

$$E = \epsilon \sigma T^4 \tag{2.48}$$

where σ is the Stefan-Boltzman constant.

2.5 Eddy Dissipation Concept (EDC)

This section is based on Ertesvågs presentation of the Eddy Dissipation Concept (EDC) in *Turbulent Flow and Combustion* [13] (in Norwegian: *Turbulent Strøyming og Forbrenning, pp. 171-189*).

The Eddy Dissipation Concept (EDC) was proposed by Magnussen and Hjertager and has been developed for decades [1][2][3]. EDC model assumes that the chemical reaction in a turbulent flow takes place where reactants and hot products are molecularly mixed. As described in Section 2.1.3, the molecular mixing occurs where the time of diffusion over the diameter of an eddy is shorter than the time an eddy takes to rotate 1/2 revolution. These locations are referred as fine structures and are also the place where TKE is dissipated into heat.

2.5.1 Energy Cascade

The EDC model is developed for RANS, which treats average values and is taken into account in the energy cascade shown in Figure 2.8. Mechanical energy, w', from the mean flow is transferred to the largest eddies with a characteristic length scale L', velocity scale u' and a frequency or strain rate $\omega' = u'/L'$. On the next level in the cascade, the characteristic frequency is $\omega'' = 2\omega'$ and length L''. Further, on the n-th level, the characteristic scales are $\omega_n = 2\omega_{n-1}$, L_n and $\omega_n = u_n/L_n$. The energy transfer continues down to the smallest level, equals Kolmogorov scale ω^*, u^* and L^*. A total

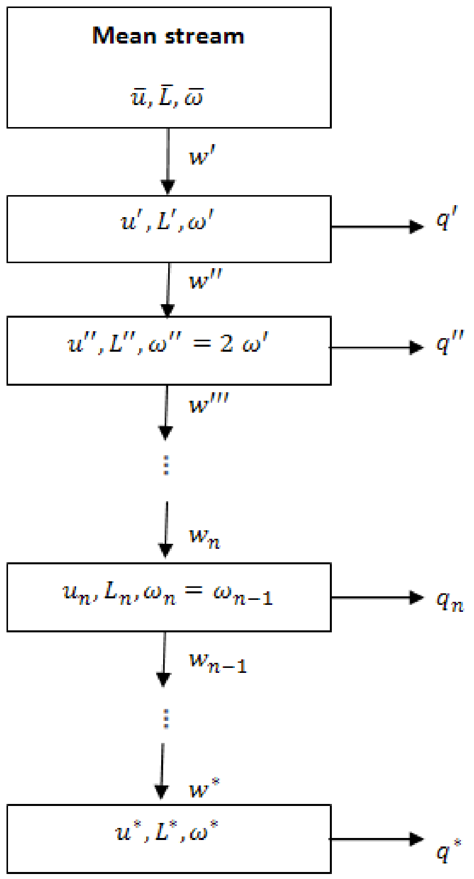


Figure 2.8: Energy cascade (adapted from Ertesvåg, 2000 [13]).

eddy dissipation for the whole cascade equal the energy transferred from the largest eddies, $\epsilon = q' + w''$. Production of TKE equals the mechanical energy transferred from the main flow to the largest eddies and is the source term in eq. (2.13) $\left(w' = \bar{\rho}\widetilde{u_i' u_j'} \frac{\partial \tilde{u}_i}{\partial x_j}\right)$.

Dissipation of TKE on the first level in the cascade is proportional to the viscosity and the strain rate:

$$w'' = \frac{3}{2} C_{D1} \omega' 2 u''^2 \tag{2.49}$$

and

$$q' = C_{D2} \nu \omega'^2 \tag{2.50}$$

C_{D1} and C_{D2} are model constants. It is assumed that w and q are equal on all levels [1]. By inserting $\omega'' = 2\omega'$ or $\omega_n = 2\omega_{n-1}$, the received and dissipated energy on n-th level yields

$$w_n = \frac{3}{2} C_{D1} \omega_n u_n^2 \tag{2.51}$$

and

$$q_n = C_{D2} \nu \omega_n^2 \tag{2.52}$$

An energy balance gives $w_n = q_n + w_{n+1}$. In turbulent flows with high Reynolds number $q_n << w_n$ and $w_n \approx w_{n+1}$ for small steps, n, and $u''^2 = \frac{1}{2}u'^2$ yields

$$w'' = \frac{3}{2}C_{D1}\omega' u'^2 = C_{D1}\omega' k \tag{2.53}$$

and

$$q' = C_{D2}\nu\omega'^2 \tag{2.54}$$

On the final step of the cascade, in the fine structure, $w^* = q^*$ and

$$w^* = \frac{3}{2}C_{D1}\omega * u^{*2} \tag{2.55}$$

and

$$q^* = C_{D2}\nu\omega^{*2} \tag{2.56}$$

At one level in the cascade, the dissipation rate is 1/4 of the dissipation rate on the level below [13]. Nearly no turbulence energy is dissipated into heat in the largest eddies and according to the model, 3/4 of the dissipation takes place in the fine structure [1]. 98% of the dissipated heat takes places in the last three steps of the cascade [13]. In flows with high Reynolds number $\epsilon \approx w''$ and $\epsilon = \frac{4}{3}q^*$, since w' is far greater than q':

$$\epsilon = w'' = \frac{3}{2}C_{D1}\frac{u'^3}{L'} \tag{2.57}$$

$$\epsilon = \frac{4}{3}q^* = \frac{4}{3}C_{D2}\nu\frac{u^{*2}}{L^{*2}} \tag{2.58}$$

A balance for last level, the fine structure is

$$\epsilon = \frac{4}{3}w^* = 2C_{D1}\frac{u^{*3}}{L^*} \tag{2.59}$$

Now, the characteristic scales for the fine structures may be expressed as

$$L^* = \frac{2}{3}\left(\frac{3C_{D2}^3}{C_{D1}^2}\right)^{1/4}\left(\frac{\nu^3}{\epsilon}\right)^{1/4} \tag{2.60}$$

$$u^* = \left(\frac{C_{D2}}{3C_{D1}^2}\right)^{1/4}(\nu\epsilon)^{1/4} \tag{2.61}$$

2.5.2 Fine Structures and Mass Exchange

The modeled ratio of fine structure mass to the total mass is

$$\gamma^* = \left(\frac{u^*}{u'}\right)^3 = \left(\frac{3C_{D2}}{4C_{D1}^2}\right)^{3/4}\left(\frac{\nu\epsilon}{k^2}\right)^{3/4} = 9.8\left(\frac{\nu\epsilon}{k^2}\right)^{3/4} \tag{2.62}$$

The ratio can be seen as a thin layer of fine structure, L^*, on a cylinder representing an eddy of diameter L', where the ratio is $\gamma^* \approx \frac{L^*}{L'}$ [13]. In between large eddies, regions of fine structures appears. The ratio of this mass to the total mass is

$$\gamma_\lambda = \frac{u^*}{u'} = \left(\frac{3C_{D2}}{4C_{D1}^2}\right)^{1/4} \left(\frac{\nu\epsilon}{k^2}\right)^{1/4} = 2.1 \left(\frac{\nu\epsilon}{k^2}\right)^{1/4} \tag{2.63}$$

Mass exchange between the fine structure to the surroundings, divided on the mass of the fine structure is expressed as

$$\dot{m}^* = 2\frac{u^*}{L^*} = \left(\frac{3}{C_{D2}}\right)^{1/2} \left(\frac{\epsilon}{\nu}\right)^{1/2} = 2.5 \left(\frac{\epsilon}{\nu}\right)^{1/2} \tag{2.64}$$

Then, the mass exchange between the fine structure to the surroundings, divided on the total mass must be $\dot{m} = \dot{m}^*\gamma^*$:

$$\dot{m} = \frac{3}{4C_{D1}} \left(\frac{12C_{D2}}{C_{D1}^2}\right)^{1/4} \left(\frac{\nu\epsilon}{K^2}\right)^{1/4} \frac{\epsilon}{k} = 24 \left(\frac{\nu\epsilon}{k^2}\right)^{1/4} \frac{\epsilon}{k} \tag{2.65}$$

The duration of mixing must be long enough that the hot products starts the reaction between fuel and oxygen. This time is given by

$$\tau^* = \frac{1}{\dot{m}^*} \tag{2.66}$$

2.5.3 Perfectly Stirred Reactor (PSR)

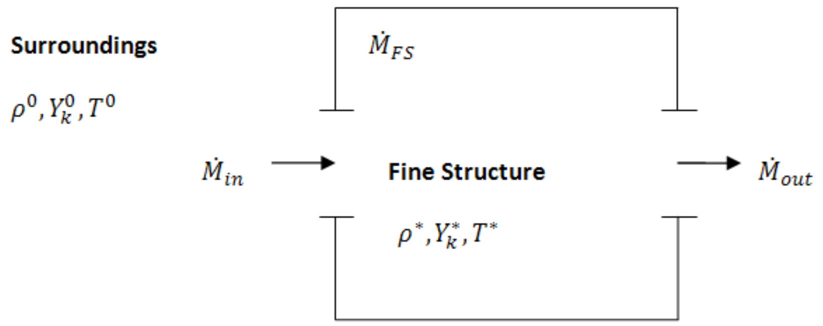


Figure 2.9: Reactor model (adapted from Ertesvåg, 2000 [13]).

The fine structures, where the reaction occurs, are treated as perfectly stirred reactor (PSR) which means that the reactants are perfectly mixed and the mass is constant at each time step [5]. The combustion in the reactor can be modeled with fast chemistry, equilibrium or chemical kinetics. Mass balance in the reactor may be expressed as

$$\dot{M}_{in}Y_k^0 - \dot{M}_{out}Y_k^* = -R_k^*\frac{M_{FS}}{\rho^*} \tag{2.67}$$

where R_k^* is the reaction rate of the fine structure, the superscripts ^{0} and * represents values for surroundings and fine structure, respectively (Figure 2.9). At stationary conditions $\dot{M}_{in} = \dot{M}_{out} = \dot{M}$. Inserting $\dot{m}^* = \frac{\dot{M}}{M_{FS}}$ in eq. (2.67) yields

$$-R_k^* = \rho^* \dot{m}^* (Y_k^0 - Y_k^*) \tag{2.68}$$

The average reaction rate is

$$\bar{R}_k = \left(R_k^* \frac{M_{FS}}{\rho^*} \right) \left(\frac{M_{tot}}{\bar{\rho}} \right)^{-1} = \frac{\bar{\rho}}{\rho^*} \gamma^* R_k^* \tag{2.69}$$

Not all of the fine structures are reacting, so a reaction fraction is given by a probability function χ:

$$\bar{R}_k = \frac{\bar{\rho}}{\rho^*} \chi \gamma^* R_k^* \tag{2.70}$$

Using $\dot{m} = \gamma^* \dot{m}^*$ and inserting eq. (2.68) in eq. (2.70) yields

$$-\bar{R}_k = \bar{\rho} \dot{m} \chi (Y_k^0 - Y_k^*) \tag{2.71}$$

As mentioned previously, RANS is treating average values. Therefore must eq. (2.71) must be expressed by an average mass fraction:

$$-\bar{R}_k = \frac{\bar{\rho} \dot{m} \chi}{1 - \gamma^* \chi} (\tilde{Y}_k - Y_k^*) \tag{2.72}$$

A detailed description of averaging is found in *Turbulent Flow and Combustion* by Ertesvåg [13]. When assuming fast chemistry, the reaction rate is

$$-\bar{R}_F = \frac{\bar{\rho} \dot{m} \chi}{1 - \gamma^* \chi} Y_{min} \tag{2.73}$$

where

$$Y_{min} = \min \left(Y_F, \frac{Y_{O_2}}{s} \right); s = \frac{W_f}{\nu_{O_2} W_{O_2}} \tag{2.74}$$

The probability function consists three parameters

$$\chi = \chi_1 \cdot \chi_2 \cdot \chi_3 \tag{2.75}$$

where χ_1 is the probability of coexistence of reactants

$$\chi_1 = \frac{\left(\tilde{Y}_{min} + \tilde{Y}_P/(1+s) \right)^2}{\left(\tilde{Y}_F + \tilde{Y}_P/(1+s) \right) \left(\tilde{Y}_{O_2}/s + \tilde{Y}_P/(1+s) \right)} \tag{2.76}$$

χ_2 the degree of heating

$$\chi_2 = \min \left[\frac{1}{\gamma_\lambda} \cdot \frac{\tilde{Y}_P/(1+s)}{\tilde{Y}_P/(1+s) + \tilde{Y}_{min}}, 1 \right] \tag{2.77}$$

and χ_3 the limitation due to lack of reactants

$$\chi_2 = \min \left[\frac{\gamma_\lambda \left(\tilde{Y}_P/(1+s) + \tilde{Y}_{min} \right)}{\tilde{Y}_{min}}, 1 \right] \tag{2.78}$$

Chapter 3

Fire Dynamics Simulator (FDS)

The Fire Dynamics Simulator (FDS) is an open source CFD code used world wide in fire engineering applications and science. For almost 25 years FDS has been developed, mainly by the National Institute of Standards and Technology (NIST) and is today accepted as industrial standard. However, the first public release was in February 2000 [4]. The main purposes of FDS are to study smoke spread, smoke venting and activation of detectors in natural building fires.

FDS calculation consists three parts: pre-processing, processing and post-processing. In the pre-processing part, a text file is written in a plain text editor where input values as boundary conditions and other initial conditions as temperature, pressure, HRR are specified. The processing part is the calculation itself, solved with FDS. In Smokeview, the post-processor, animations and images of output values can be analyzed. Output values may also be exported to Excel, MatLab or similar softwares. Third party softwares exist for pre-processing and/or post-processing. The most popular commercial software is PyroSim.

FDS solves numerically the Navier-Stokes equations with low Mach number approximation, $Ma < 0.3$, on a (uniform) cartesian grid. This limits simulations to thermally-driven flows with low pressure differences. The equations are solved explicitly in second order central difference schemes in space and time [25]. Reynolds stresses are closed either with Deardorff SGS Model (current default), Smagorinsky SGS model (constant or dynamic coefficient) or Vreman (in the unoffical version 6).

3.1 Local Heat Release Rate (HRR) and Reaction Rate

The reaction rate in a combustion process is assumed to be limited by either the chemical reaction time (finite-rate reaction) or by mixing time of fuel, oxygen and hot products (infinitely fast reaction). When using LES the grid resolution is too coarse to compute the reaction rate directly because the flame sheet is much thinner than a grid cell [25]. I.e combustion is a subgrid phenomenon and need to be modeled. If the condition in a grid cell meets certain criteria and the reactants of the reaction are present, the local

HRR is modeled as

$$\dot{Q}_c = \rho \min\left(Y_F, \frac{Y_{O_2}}{s}, \beta\frac{Y_P}{1+s}\right)\left(1 - e^{-\delta t/\tau}\right)\Delta H_c \tag{3.1}$$

if simple chemistry (one-step reaction) and eddy dissipation i specified in the input file, else HRR is

$$\dot{Q}_c = \rho \min\left(Y_F, \frac{Y_{O_2}}{s}, \beta\frac{Y_P}{1+s}\right)\left(\frac{1}{\tau}\right)\Delta H_c \tag{3.2}$$

where Y_F, Y_{O_2} and Y_P are mass fractions of fuel, oxygen and products, respectively. The empirical parameter β is equal 1, δt is the time step and $s = \frac{W_f}{\nu_{O_2}W_{O_2}}$. τ is a characteristic mixing time given by

$$\tau = \max\left(\tau_{chem}, \min\left(\tau_d, \tau_u, \tau_g, \tau_{flame}\right)\right) \tag{3.3}$$

where the diffusion time scale, turbulent time scale and acceleration time scale are

$$\tau_d = \frac{Sc_t \rho \Delta^2}{\mu}; \tau_u = \frac{\Delta}{\sqrt{2k_{sgs}}}; \tau_g = \sqrt{\frac{2\Delta}{g}} \tag{3.4}$$

Sc is the dimensionless Schmidt number, defined as the ratio of viscous diffusion rate and molecular diffusion rate (default is 0.5 [25]). The subgrid kinetic energy, k_{sgs}, is calculated by integrating a model Kolmogorov spectrum [25]. τ_g is acceleration time scale and is the time required to travel a distance Δ under a constant acceleration, $g = 9.81$ m/s^2. The flame time scale (large), τ_{flame}, and the chemistry time scale (small), τ_{chem} may be specified by the user. They are however, rarely the limiting mixing time. The simple mixing model used in FDS is grid depended and will overestimate the HRR when a too coarse grid resolution is applied. An upper bound on the total local heat release rate per unit volume (HRRPUV) is set to 2500 kW/m^3.

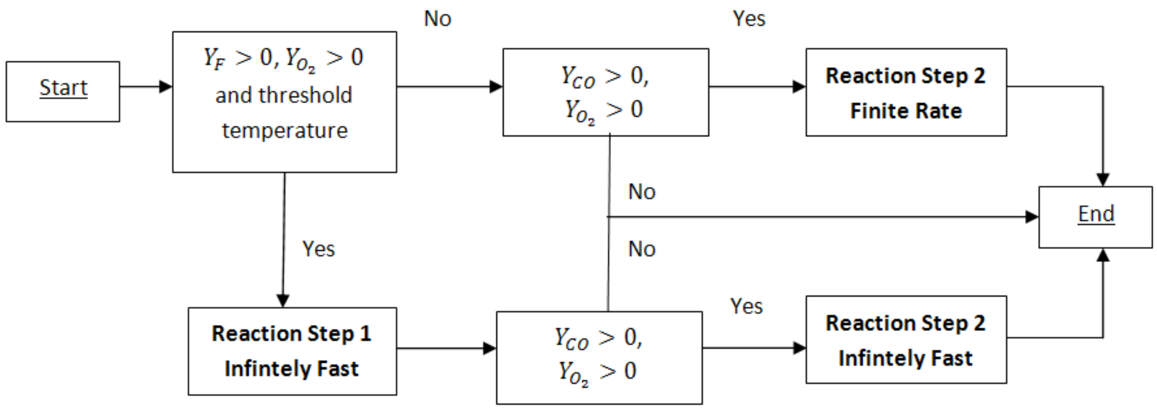

Start
$Y_F > 0, Y_{O_2} > 0$ and threshold temperature
No
Yes
$Y_{CO} > 0$, $Y_{O_2} > 0$
Yes
Reaction Step 2 Finite Rate
No
No
End
Reaction Step 1 Infintely Fast
$Y_{CO} > 0$, $Y_{O_2} > 0$
Yes
Reaction Step 2 Infintely Fast

Figure 3.1: A description of how the reaction rate is computed in FDS.

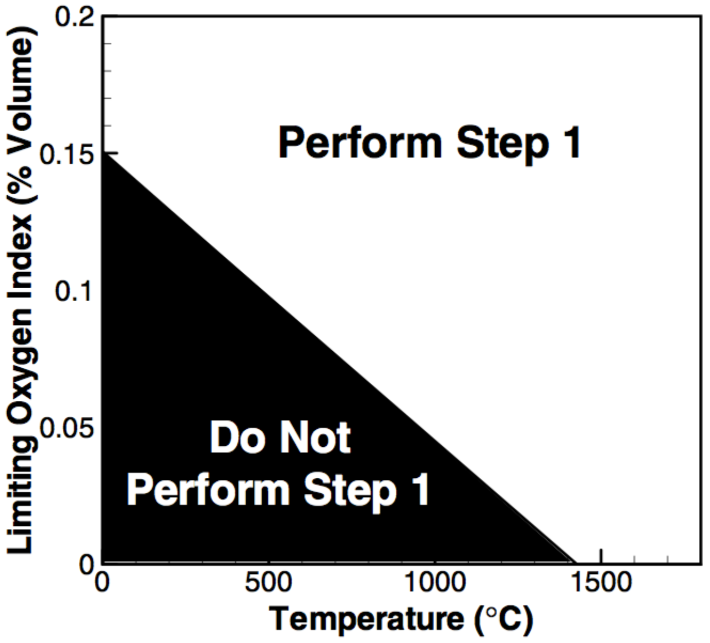


Figure 3.2: The threshold temperature for a given oxygen concentration [26].

When assuming finite rate reaction, the reaction rate i computed by Arrhenius rate, used for DNS calculations and post-flame measurements. A detailed description is found in FDS Technical Reference Guide [25].

For the two-step reaction, fuel and oxygen are reacting infinitely fast (eq. (3.1)) forming CO, soot and other products if the temperature for a given oxygen concentration fulfills Figure 3.2. Further, CO is reacting infinitely fast with oxygen if any oxygen is left in the grid cell after the first reaction. If no fuel is traced in the first reaction or the temperature is too low to support combustion, but CO is present, a finite rate assumed. A visual description is seen in Figure 3.1.

3.2 Mulit-Parameter Mixture Fraction

With a single parameter mixture fraction model information of the completeness of the reaction is not stored [26]. In FDS v4 a single parameter mixture fraction, Z, given in eq. (3.5) was used. The scalar represents the original mass of fuel before the combustion.

$$Z = Y_F + \frac{W_F}{xW_{CO_s}} Y_{CO_2} + \frac{W_f}{xW_C O} Y_{CO} + \frac{W_F}{xW_s} Y s \tag{3.5}$$

In well-ventilated fires the single scalar approach is sufficient. In other cases as extinctions and under-ventilated fires resulting in incomplete reaction, a single parameter is not sufficient. In order to get information of CO and CO_2 production, multi-parameter mixture fraction model is necessary.

FDS supports

- two parameters single-step reaction: Z_0, Z_1 and Z_2
- three parameters two-step reaction: Z_0, Z_1, Z_2 and Z_3

where $Z = \sum_\alpha Z_\alpha$. For a single step reaction fuel is reacting with oxygen and do not allow post combustion of CO. A fixed amount of CO is formed. In contrast, a two-step approach allows post combustion of CO within a hot upper layer in under-ventilated fires, post flame combustion or CO production due to fire suppression [25][26]. Note that a yield of CO (ν_{CO}) is still needed to be specified when using a two-step reaction. Also the yields of CO_2 and soot (ν_{CO_2}, ν_s) in eq. (3.25) and (3.6) must be specified by the user, for either a complete or incomplete reaction.

3.2.1 Two parameter single-step reaction

For a single-step approach, the reaction is

$$C_xH_yO_zN_a + \nu'_{O_2}O_2 \rightarrow (\nu_{CO_2}CO_2 + \nu_{H_2O}H_2O + \nu_{CO})CO + \nu_s S + \nu_{n_2}N_2 \tag{3.6}$$

where x is the number of carbon atoms, y the number of hydrogen atoms, z the number of oxygen atoms and a the number of nitrogen atoms. A general form of the reaction is

$$Fuel + \nu_{air} \rightarrow Products \tag{3.7}$$

The species are Z_0 for air, Z_1 for fuel and Z_2 for products. Z_1 and Z_2 are tracked explicitly while Z_0 is tracked implicitly as the background species. The mass fractions are given as [25]:
Z_0: Air

$$Y_{N_2}(Z_0) = Y^\infty_{N_2} \tag{3.8}$$
$$Y_{O_2}(Z_0) = Y^\infty_{O_2} \tag{3.9}$$
$$Y_{CO_2}(Z_0) = Y^\infty_{CO_2} \tag{3.10}$$
$$Y_{H_2O}(Z_0) = Y^\infty_{H_2O} \tag{3.11}$$

Z_1: Fuel

$$Y_F(Z_1) = Y_F \tag{3.12}$$

Z_2: Products

$$Y_{N_2}(Z_2) = \frac{\nu_{air}W_{air}Y^\infty_{N_2} + \nu_{N_2}W_{N_2}}{W_F + \nu_{air}W_{air}} \tag{3.13}$$
$$Y_{CO_2}(Z_2) = \frac{\nu_{air}W_{air}Y^\infty_{CO_2} + \nu_{CO_2}W_{CO_2}}{W_F + \nu_{air}W_{air}} \tag{3.14}$$
$$Y_{H_2O}(Z_2) = \frac{\nu_{air}W_{air}Y^\infty_{H_2O} + \nu_{H_2O}W_{H_2O}}{W_F + \nu_{air}W_{air}} \tag{3.15}$$
$$Y_{CO}(Z_2) = \frac{\nu_{CO}W_{CO}}{W_F + \nu_{air}W_{air}} \tag{3.16}$$
$$Y_S(Z_2) = \frac{\nu_S W_S}{W_F + \nu_{air}W_{air}} \tag{3.17}$$

The species yields given as:

$$Y_\alpha(Z_0, Z_1, Z_2) = Y_\alpha(Z_0)(1 - Z_1 - Z_2) + Y_\alpha(Z_1)Z_1 + Y_\alpha(Z_2)Z_2 \tag{3.18}$$

Stoichiometric coefficients in Z_2 are:

$$\nu_{N_2} = \frac{a}{2} \tag{3.19}$$

$$\nu_{O_2} = \nu_{CO_2} + \frac{\nu_{CO} + \nu_{H_2O} - z}{2} \tag{3.20}$$

$$\nu_{CO_2} = x - \nu_{CO} - (1 - X_H)\nu_S \tag{3.21}$$

$$\nu_{H_2O} = \frac{y}{2} - X_H \nu_S \tag{3.22}$$

$$\nu_{CO} = \frac{W_F}{W_{CO}} y_{CO} \tag{3.23}$$

$$\nu_S = \frac{W_F}{W_S} y_S \tag{3.24}$$

3.2.2 Three parameter two-step reaction

The reactions for a to step reaction are

$$C_xH_yO_zN_a + \nu'_{O_2}O_2 \to \nu_{H_2O}H_2O + (\nu'_{CO} + \nu_{CO})CO + \nu_s S + \nu_{n_2}N_2 \tag{3.25}$$

$$\nu'_{CO}\left[CO + \frac{1}{2}O_2 \to CO_2\right] \tag{3.26}$$

The species in the two-step approach are Z_0 for air, Z_1 for fuel, Z_2 for products in the incomplete reaction one and Z_2 for products in the complete reaction two. A simplified way of writing the the two step reaction is

$$Fuel + \nu_{Air,1}Air \to Incomplete\ Products \tag{3.27}$$

$$Incomplete\ Products + \nu_{Air,2}Air \to Complete\ Products \tag{3.28}$$

The species yields are:
Z_0: Air

$$Y_{N_2}(Z_0) = Y^\infty_{N_2} \tag{3.29}$$

$$Y_{O_2}(Z_0) = Y^\infty_{O_2} \tag{3.30}$$

$$Y_{CO_2}(Z_0) = Y^\infty_{CO_2} \tag{3.31}$$

$$Y_{H_2O}(Z_0) = Y^\infty_{H_2O} \tag{3.32}$$

Z_1: Fuel

$$Y_F(Z_1) = Y_F \tag{3.33}$$

Z_2: Products of Incomplete Reaction

$$\begin{aligned}
Y_{N_2}(Z_2) &= \frac{\nu_{air,1} W_{air} Y_{N_2}^{\infty} + \nu_{N_2} W_{N_2}}{W_F + \nu_{air} W_{air}} && (3.34)\\
Y_{CO_2}(Z_2) &= \frac{\nu_{air} W_{air} Y_{CO_2}^{\infty}}{W_F + \nu_{air} W_{air}} && (3.35)\\
Y_{H_2O}(Z_2) &= \frac{\nu_{air} W_{air} Y_{H_2O}^{\infty} + \nu_{H_2O} W_{H_2O}}{W_F + \nu_{air} W_{air}} && (3.36)\\
Y_{CO}(Z_2) &= \frac{\nu_{CO'} W_{CO}}{W_F + \nu_{air} W_{air}} && (3.37)\\
Y_S(Z_2) &= \frac{\nu_S W_S}{W_F + \nu_{air} W_{air}} && (3.38)
\end{aligned}$$

Z_3: Products of Complete Reaction

$$\begin{aligned}
Y_{N_2}(Z_3) &= \frac{\nu_{air,2} W_{air} Y_{N_2}^{\infty} + \nu_{N_2} W_{N_2}}{W_F + \nu_{air} W_{air}} && (3.39)\\
Y_{CO_2}(Z_3) &= \frac{\nu_{air} W_{air} Y_{CO_2}^{\infty}}{W_F + \nu_{air} W_{air}} && (3.40)\\
Y_{H_2O}(Z_3) &= \frac{\nu_{air} W_{air} Y_{H_2O}^{\infty} + \nu_{H_2O} W_{H_2O}}{W_F + \nu_{air} W_{air}} && (3.41)\\
Y_{CO}(Z_3) &= \frac{\nu_{CO} W_{CO}}{W_F + \nu_{air} W_{air}} && (3.42)\\
Y_S(Z_3) &= \frac{\nu_S W_S}{W_F + \nu_{air} W_{air}} && (3.43)
\end{aligned}$$

The stoichiometric coefficients are:

$$\begin{aligned}
\nu_{N_2} &= \frac{a}{2} && (3.44)\\
\nu'_{O_2} &= \frac{\nu'_{CO} + \nu_{H_2O} - z}{2} && (3.45)\\
\nu_{O_2} &= \nu_{CO_2} + \frac{\nu_{CO} + \nu_{H_2O} - z}{2} && (3.46)\\
\nu_{CO_2} &= x - (1 - X_H)\nu_S && (3.47)\\
\nu_M &= b && (3.48)\\
\nu_{H_2O} &= \frac{y}{2} - X_H \nu_S && (3.49)\\
\nu'_{CO} &= x - \nu_{CO} - (1 - X_H)\nu_s && (3.50)\\
\nu'_{CO} &= \frac{W_F}{W_{CO}} y_{CO} && (3.51)\\
\nu_S &= \frac{W_F}{W_S} y_S && (3.52)
\end{aligned}$$

3.3 Extinction Criteria

The model for local extinction in FDS consists two parts. First, the local temperature in a grid cell has to be above auto-ignition temperature to the fuel. Second, the local temperature rise must exceed the lower flame temperature, T_{LFT}, from the energy released when maximum limiting available amount of fuel or oxygen, in terms of fuel are consumed:

$$\Delta Z_{air} h_{air}(T) + \Delta Z_F (h_F(T) + \Delta H_F) > \Delta Z_{air} h_{air}(T_{LFT}) + \Delta Z_F h_F(T_{LFT}) \tag{3.53}$$

3.4 Grid Resolution

To find the correct grid resolution is not straightforward. It depends on input values in the calculation and no accepted manual to this exist in the CFD community. Bjarne P. Husted *et al.* recommended that the grid cell size should be in such order that 90-99% of the turbulent energy is solved on the grid points [27]. In FDS User's Manual a somewhat lower value on at least 80% is recommended. Generally, the correct grid cell size is found by simulating the scenario with relative coarse grid and then gradually refine it until only minor differences in the results are observed. This procedure is called a grid sensitivity analysis. For low Mach number LES approximation, D^*/dx is a measure on how well the flow field is resolved. The ratio D^*/dx may also be used to predict the correct grid cell size. The non-dimensional fire diameter is

$$D^* = \left(\frac{\dot{Q}}{\rho_\infty c_p T_\infty \sqrt{g}} \right)^{\frac{2}{3}} \tag{3.54}$$

and dx is the grid cell size. The ratio D^*/dx should be between 4 and 16 [25][4]. T_∞ is the ambient temperature and ρ_∞ is the ambient density. Since CFD modeling is time consuming, it is desirable to not use a finer grid than necessary to obtain satisfactory results. It is worth noting that doubling the number of nodes in each direction, reduces the discretization error by a factor of 4. Furthermore, the computing time increases by a factor of 16 (a factor of 2 for the temporal and each spatial dimension) [28]. To enhance the calculation with respect to time, the domain can be divided in several grids and *Multi Processor Interface* (MPI) may be applied to calculate the grids in parallel.

Chapter 4

Implementation of LES-EDC in FDS

4.1 LES-EDC

Extension of EDC to LES is not a straightforward task and requires customizing. In LES, partial turbulence cascading or no cascading occurs. However, the fine structure regions are calculated based on full energy cascading in each grid cell in RANS-EDC. Panjwani *et al.* proposed two approaches for formulating the fine structure regions [5][6]. In contrast to RANS, the turbulent kinetic energy and dissipation are rarely computed explicitly (i.e no conservation) in LES. Thus, the fine structure regions are formulated from the subgrid viscosity. The first approach is based on volume fraction of small structures, the second on inhomogenious distribution of the fine structure regions, both ending up with the expression

$$\gamma_\lambda = C_{LES}\left(\frac{\nu}{\nu_{sgs}}\right)^{\frac{1}{4}} \tag{4.1}$$

where the subgrid viscosity, ν_{sgs}, is tracked from the turbulence model. In case of larger fine structure velocity than subgrid velocity , $\gamma_\lambda < 1$ is used [5]. The proposed EDC-LES model assumes that the fine structures are localized in nearly constant energy regions so that $\gamma^* = \gamma_\lambda^2$. Furthermore, the time scale is reformulated [29]:

$$\tau^* = \frac{1}{|\bar{S}|} \tag{4.2}$$

where $\bar{S}$ is the strain rate. The reaction rate is computed as

$$\omega_k = \frac{\gamma_\lambda^2 \chi}{\tau^*}\left(\tilde{Y}_k^0 - \tilde{Y}_k^*\right) \tag{4.3}$$

where Y_k^0 is the surrounding mass fraction and Y_k^* the mass fraction of the fine structures, see Figure 2.9. In terms of limiting concentration of fuel or oxygen, the reaction rate is

$$\omega_k = \frac{\chi}{\tau^*(1-\chi\gamma_\lambda^2)} Y_{min} \tag{4.4}$$

where

$$Y_{min} = \min\left(Y_F, \frac{Y_{O_2}}{s}\right); s = \frac{W_f}{\nu_{O_2} W_{O_2}} \tag{4.5}$$

The probability function, χ, is computed as fo RANS-EDC given in eq. (2.76) - (2.78).

4.2 Numerical Procedure

FDS is a non-commercial code written in FORTRAN 90/95 and is available on the website `http://code.google.com/p/fds-smv/`. Minor releases or subversions of FDS are accessed through a Subversion Client (SVN), while major releases are published on NIST's official website (`http://fire.nist.gov/fds/`).

The FDS code is divided in modules and subroutines as CFD in general. In this thesis the subroutine "Eddy Dissipation" (infinitely fast reaction, eq. (3.1)) in module "Fire" (fire.f90 Appendix B.1) is substituted with the EDC combustion model modified for LES described in section 4.1. In addition, other suroutines were adapted to changes implemented in the combustion model. A schematic diagram of the subroutines in the module "Fire" is presented in Figure 4.1.

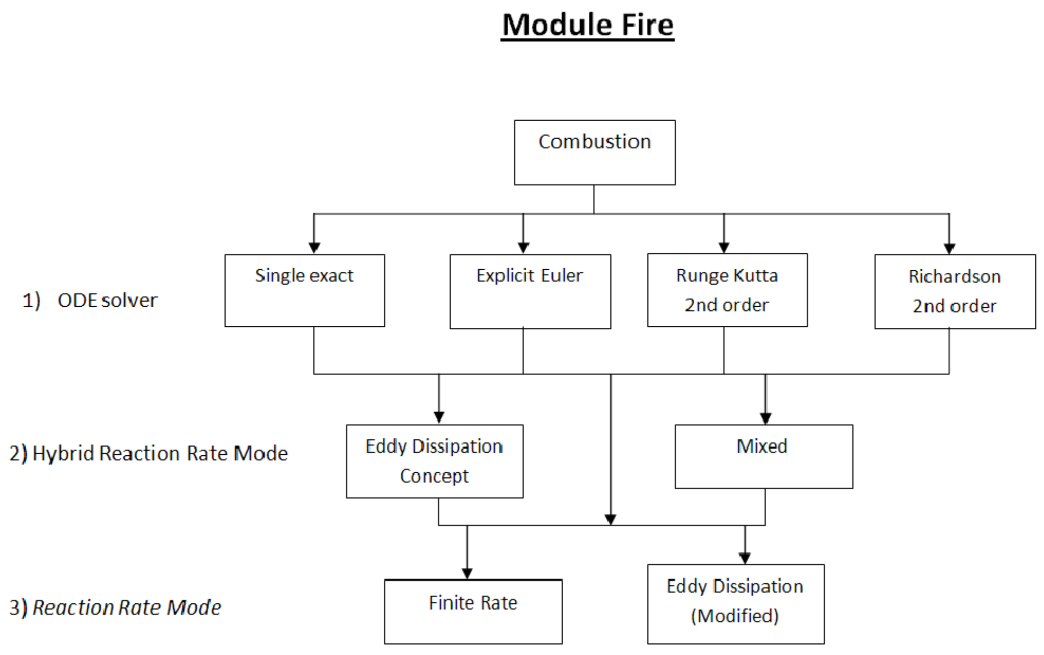


Figure 4.1: Numerical Procedure in FDS combustion model (fire.f90 in Appendix B.1).

- **Combustion:** First, the upper HRR in the grid cell is calculated, set to 2500 kW/m^3. If reactants are present, an ordinary differential equation (ODE) solver is chosen. The temperature is calculated by the energy released from the combustion, which is reported back from the ODE solver. Mixture fractions are also reported back from the ODE solver before the divergence term is updated.

- **ODE solver:** If simple chemistry (single-step reaction) and "Eddy Dissipation" reaction rate is specified in the input file, the "Single Exact" ODE solver is chosen. Otherwise, "Explicit Euler" is chosen. Use of other ODE solvers must be specified in the input file. The ODE solver reads the number of reaction steps before finding the limiting reactant in terms of fuel. The mixing time is reported from the reaction rate mode and HRR is calculated from eq. (3.1) in the "Single Exact" solver or eq. (3.2) for the other solvers. Then fuel consumption is calculated.

- **Hybrid Reaction Rate Mode:** The "Mixed" mode is following the description in Figure 3.1 and 3.2. For DNS calculations, finite rate reaction is assumed. "Eddy_Dissipation_Concept" is calculating both "Finite Rate" (RATE_CONSTANT_FR) and "Eddy Dissipation" (RATE_CONSTANT_ED). The reaction rate is then

  $(RATE\_CONSTANT\_ED \times RATE\_CONSTANT\_FR)/(RATE\_CONSTANT\_ED + RATE\_CONSTANT\_FR)$.

  "Eddy Dissipation" is supporting co-existing of fuel and oxygen in the same grid cell by a function called extinction, see section 3.3.

4.3 The Implemented Code

The major part of the changes are done in the module "fire.f90", under "Eddy Dissipation":

- First the kinematic viscosity is calculated, $\nu = \mu/\rho$, because it is not a global variable in FDS.
- The fine structure is computed by eq. (4.1), an upper value value is set $\gamma_\lambda < 1$.
- Mass fractions for oxygen and fuel are tracked from the subroutine GET_MASS_FRACTION, and mass fraction for products is found from the relation $Y_{products} = 1 - (Y_{O_2} + Y_{fuel})$.
- Stoichiometric coefficient is computed.
- The probability function is computed by eq. (2.75), (2.76), (2.77) and (2.78).
- Rate constant is computed by eq. (4.3).

```
!The Eddy Dissipation Consept (EDC) Combustion Model (
    by Hjertager and Magnussen) for LES proposed by
    Balram et al.

NU = MU(I,J,K)/RHO(I,J,K)

C_LES = 0.15_EB

GAMMA_LAMBDA = C_LES*(NU/NU_EDDY(I,J,K))**0.25_EB
IF(GAMMA_LAMBDA > 1._EB)THEN
   GAMMA_LAMBDA = 1._EB
END IF

CALL GET_MASS_FRACTION(ZZ_GET,FUEL_INDEX,Y_FUEL) !
    ADDED
CALL GET_MASS_FRACTION(ZZ_GET,O2_INDEX,Y_O2) !ADDED
Y_PRODUCT = 1._EB - (Y_FUEL + Y_O2)

S = SPECIES(FUEL_INDEX)%MW/(RN%NU_O2*SPECIES(O2_INDEX)
    %MW)
Y_O2 = Y_O2/S
Y_PRODUCT = Y_PRODUCT/(1._EB + S)
CHI_1 = ((YY_F_LIM + Y_PRODUCT)**2)/((Y_FUEL +
    Y_PRODUCT)*(Y_O2 + Y_PRODUCT))
CHI_2 = MIN(Y_PRODUCT/(GAMMA_LAMBDA*(YY_F_LIM +
    Y_PRODUCT)),1._EB)
CHI_3 = MIN(GAMMA_LAMBDA*(YY_F_LIM + Y_PRODUCT)/
    YY_F_LIM,1._EB)
CHI = CHI_1*CHI_2*CHI_3

RATE_CONSTANT = YY_F_LIM*CHI/(MIX_TIME(I,J,K)*(1._EB -
     CHI*GAMMA_LAMBDA**2))
!RATE_CONSTANT = YY_F_LIM*CHI*GAMMA_LAMBDA**2/(
    MIX_TIME(I,J,K)*(1._EB - CHI*GAMMA_LAMBDA**2))
```

The mixing time (eq. 4.2) is computed further up in the module. Other changes are done in modules ”velo.f90”, ”mesh.f90” and ”init.f90”. Since the strain rate is only computed in the Smagorinsky turbulence model (eq. (2.18)), the subroutine COMPUTE_STRAIN_RATE is added in the subroutine COMPUTE_VISCOSITY for the Deardorff (eq. (2.19)) and Vreman (eq. (2.20)) turbulence model in ”velo.f90”. The strain rate and the subgrid viscosity are made global variables by modifications in ”mesh.f90” and ”init.f90”. Furthermore, parts of the code are commented out, so that they do not conflict with the implemented part. The ODE solve ”Single Exact” is adapted to eq. (3.1) only, and are for that reason substituted by the "Explicit Euler". Limiting mass fraction in terms of fuel, Y_{min}, is part of the original code. The module fire.f90 is found in its full length in Appendix B.1. Other modified subroutines are found in Appendix B whiles their full length is found at `http://code.google.com/p/fds-smv/`.

Chapter 5

Validation of FDS-EDC

An important aspect when implementing a new model in a CFD model is to validate this. Validation can be done by comparing simulations with experiments or other numerical tools that computes the exact physics instead of modeling it (typically DNS). In some cases CFD models are validated with flows which can be solved analytically, e.g velocity profile in Couette flow.

The Standard Guide for Evaluating the Predictive Capability of Deterministic Fire Models ASTM E 1355[30] defines verification as

-The process of determining the degree to which a calculation method is an accurate representation of the real world from the perspective of the intended uses of the calculation method.

and verification as

- The process of determining that the implementation of a calculation method accurately represents the developer's conceptual description of the calculation method and the solution to the calculation method.

In FDS verification guide it is putted in a simple way and stated that verification is to check the math and validation to check the physics [31]. The choice of validation cases must be within the limits of CFD code and related to its purpose. In the matter of combustion model validation the quantities must be related to the local HRR. The energy released in the combustion increases the temperature, and through thermal expansion and buoyancy caused by density differences the flow is forced in an upward direction. The type of fire must equal those FDS is limited to and about the same size. There are several fire scenarios to consider and many ways to validate a CFD code. The selected scenarios in this thesis are identical to some of those in FDS validation guide [32], so that results in this thesis may in further work be compared with earlier and future versions of FDS. Based on the given arguments McCaffery centerline velocity and temperature profile [24], Heskestad flame height correlation [23] and velocity profile in

Sandia plume experiments [33][34] are considered in this thesis. Temperature, velocity and flame height are important quantities for flames and are all influenced by local HRR which is calculated in the implemented code. Flame height validation is a way to check if the fuel consumed over the correct travelled distance while the center line temperature and velocity profile is a way to check the local fuel consumption is correct. Even though all the fuel is consumed after traveling the distance equal the flame height the local fuel consumption do not necessarily has to be correct. Correct modeling of the lower flame region is crucial to be able to model a flame correctly. The lower flame region, called the persistent flame, is where the flow is accelerating. Above, in the intermittent region, the flow is nearly constant. An over prediction of HRR in the lower region leads to an over prediction of the velocity, hence the height of the persistent region decreases. Furthermore, the temperature is over predicted in the lower region and the flame height under predicted. An under prediction of HRR in the persistent region leads to the opposite. By velocity profiles from Sandia experiments the persistent flame region may be studied in detail. In Section 6.1, information regarding the philosophy of arranging experiments for CFD validation is found.

In total, 720 simulations are presented in this thesis; 480 simulations for the Heskestad correlation, 150 simulations for the McCaffery correlation and 90 simulations for the Sandia plume experiments. All simulations were run with FDS v.6, SVN revision number 10231 on six different computers:

- a) 2 x Intel(R) Xenon(R) CPU X5690 Hexa Core 3.47 GHz, (48 GB memory)
- b) 2 x Intel(R) Xenon(R) CPU X5690 Hexa Core 3.47 GHz, (48 GB memory)
- c) Intel(R) Xenon(R) CPU X5570 Quad Core 2.93 GHz, (12 GB memory)
- d) Intel(R) Xenon(TM) CPU Dual Core 3.0 GHz, (4 GB memory)
- e) 2 x Intel(R) Xenon(R) CPU E5540 Quad Core 2.53 GHz, (24 GB memory)
- f) 2 x Intel(R) Xenon(R) CPU E5630 Quad Core 2.53 GHz, (8 GB memory)

The Sandia plume experiments were simulated with 4 CPU cores with computer a) and b) to be able to compare CPU clock times[1]. Because of time constraint, results for the original version FDS v.6 (only referred as FDS6) were only compared with Deardorff turbulence model. Deardorff was chosen since that is the default turbulence model in the unofficial version 6 of FDS. Input for all simulations were limited to a two parameters mixture fraction with a single-step reactions for infinitely fast chemistry.

Example of input files are found in Appendix A. All input files in its full length are found at `http://code.google.com/p/fds-smv/`. Results are presented with three different C_{LES} and three different grid resolutions.

[1]Test 17 with Vreman turbulence model ($dx = 1.5$ cm) was simulated with 12 CPU cores and stopped at 15 seconds because of computer crash and time constraint in the final stages of the project.

5.1 McCaffery's Plume Correlation, Velocity and Temperature Profiles

The McCaffery case is simulated with a 30 cm x 30 cm methane burner with HRR of 14 kW, 22 kW, 33 kW, 45 kW and 57 kW. Three different grids (30x30x100, 60x60x200 and 40x40x100) are applied with varying computational domain size such that $D^*/dx = 5$, $D^*/dx = 10$ and $D^*/dx = 20$. This corresponds a range of dimensionless heat release from $Q^* = 0.005$ to $Q^* = 0.02$. The centerline velocity and temperature profiles are average over the 30 seconds simulated. Ambient conditions are 20°C and atmospheric pressure. Results are presented for Deardorff, Vreman and (dynamic) Smagorinsky turbulence models with C_{LES} of 0.005, 0.01 and 0.015. An example of input file is found in Appendix A.5.

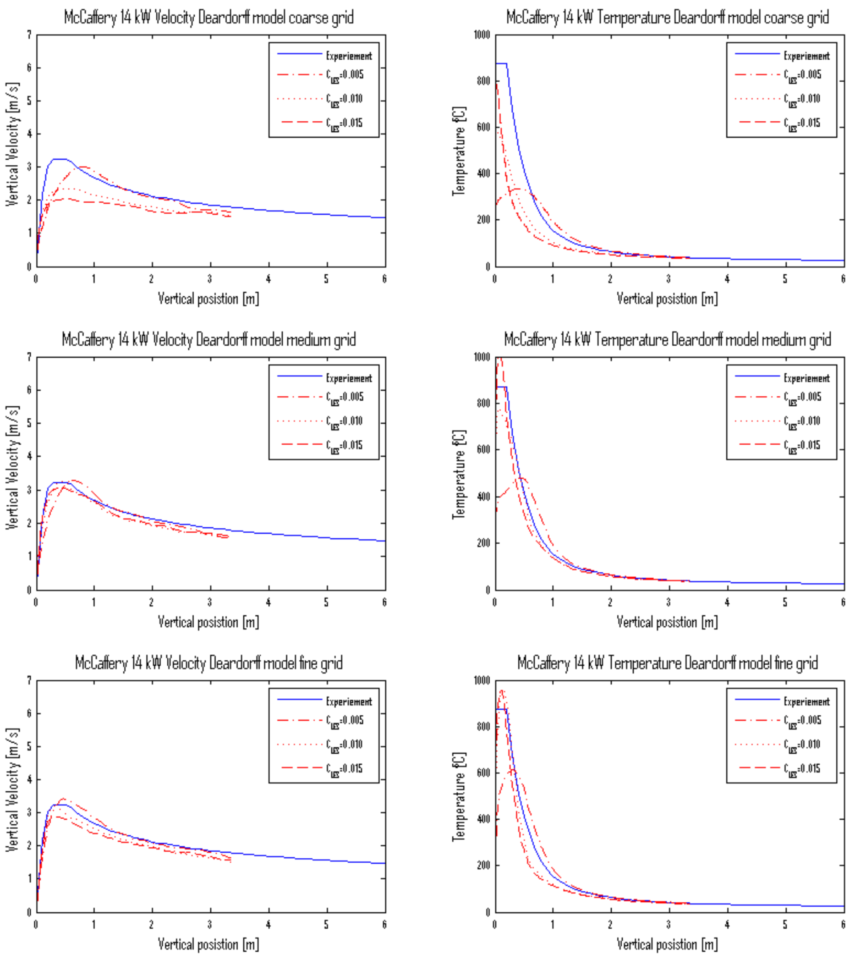


Figure 5.1: McCaffery correlation 14 kW, centerline velocity (left side) and temperature (right side) profiles with Deardorff turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

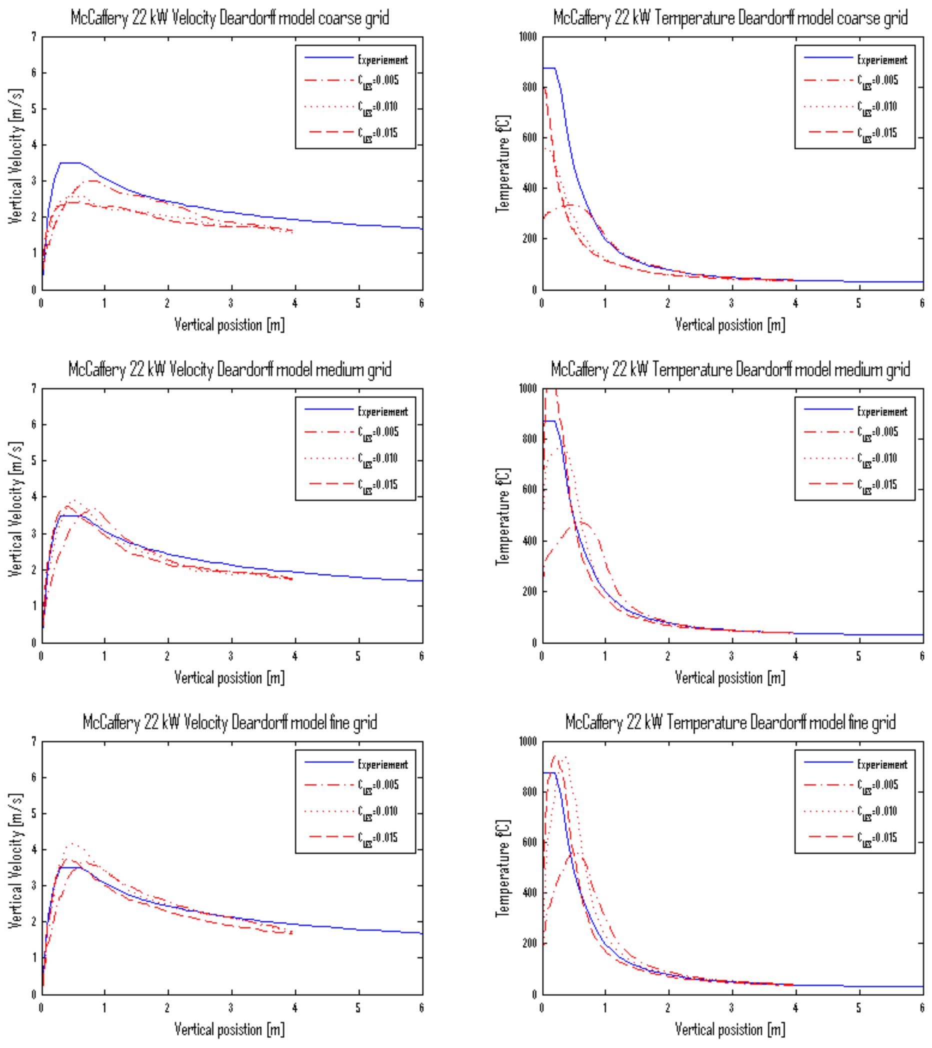


Figure 5.2: McCaffery correlation 22 kW, centerline velocity (left side) and temperature (right side) profiles with Deardorff turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

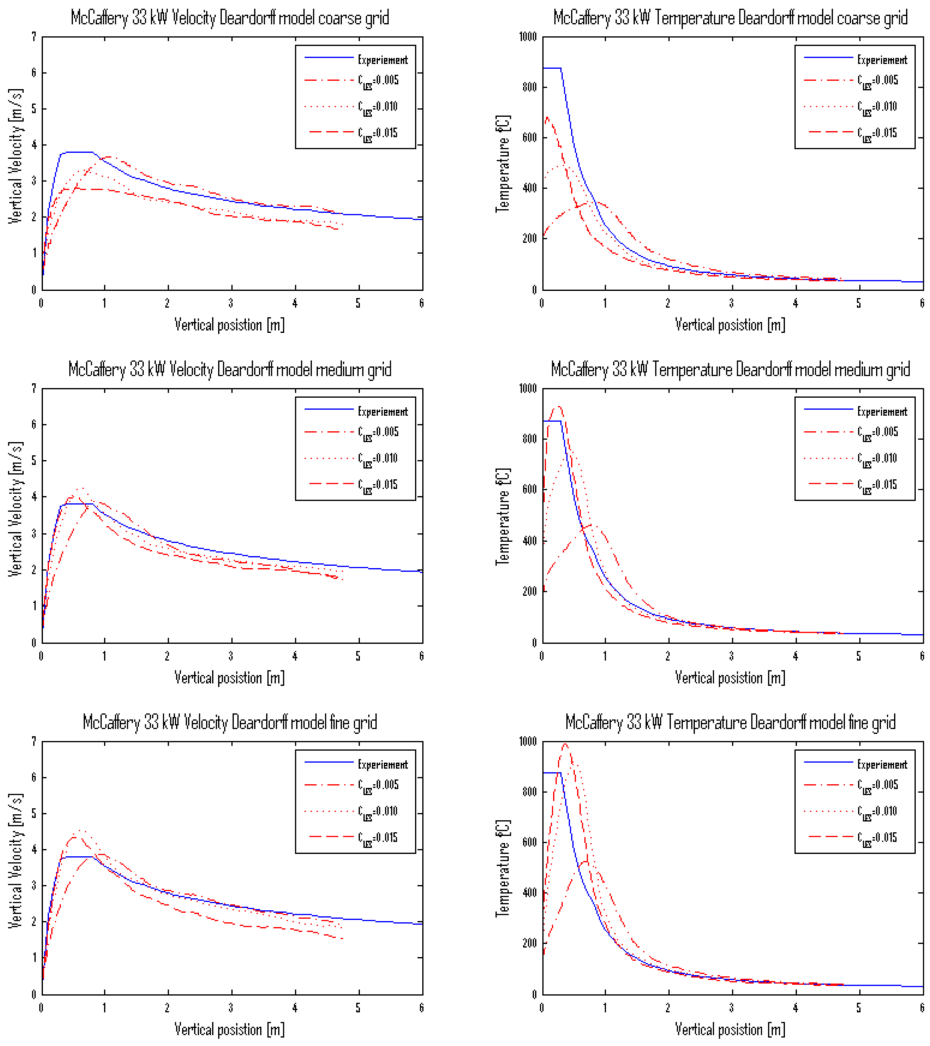


Figure 5.3: McCaffery correlation 33 kW, centerline velocity (left side) and temperature (right side) profiles with Deardorff turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

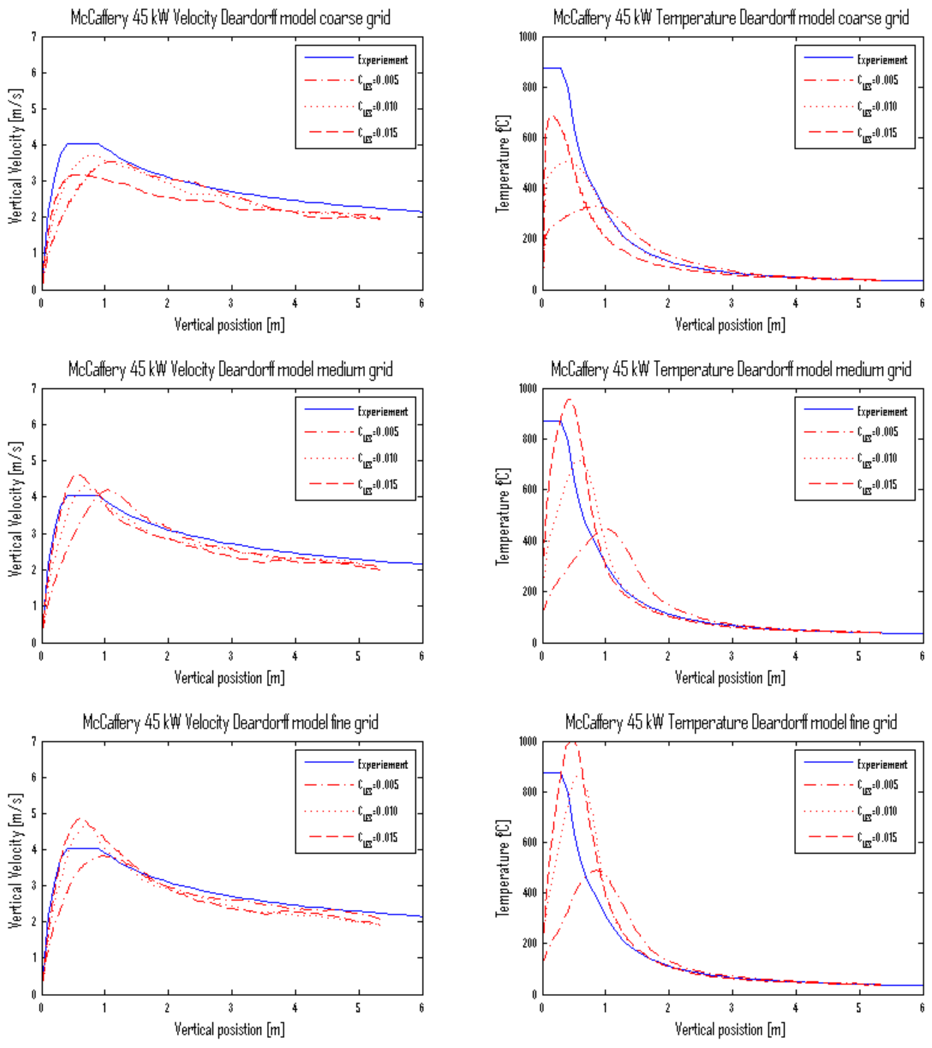


Figure 5.4: McCaffery correlation 45 kW, centerline velocity (left side) and temperature (right side) profiles with Deardorff turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

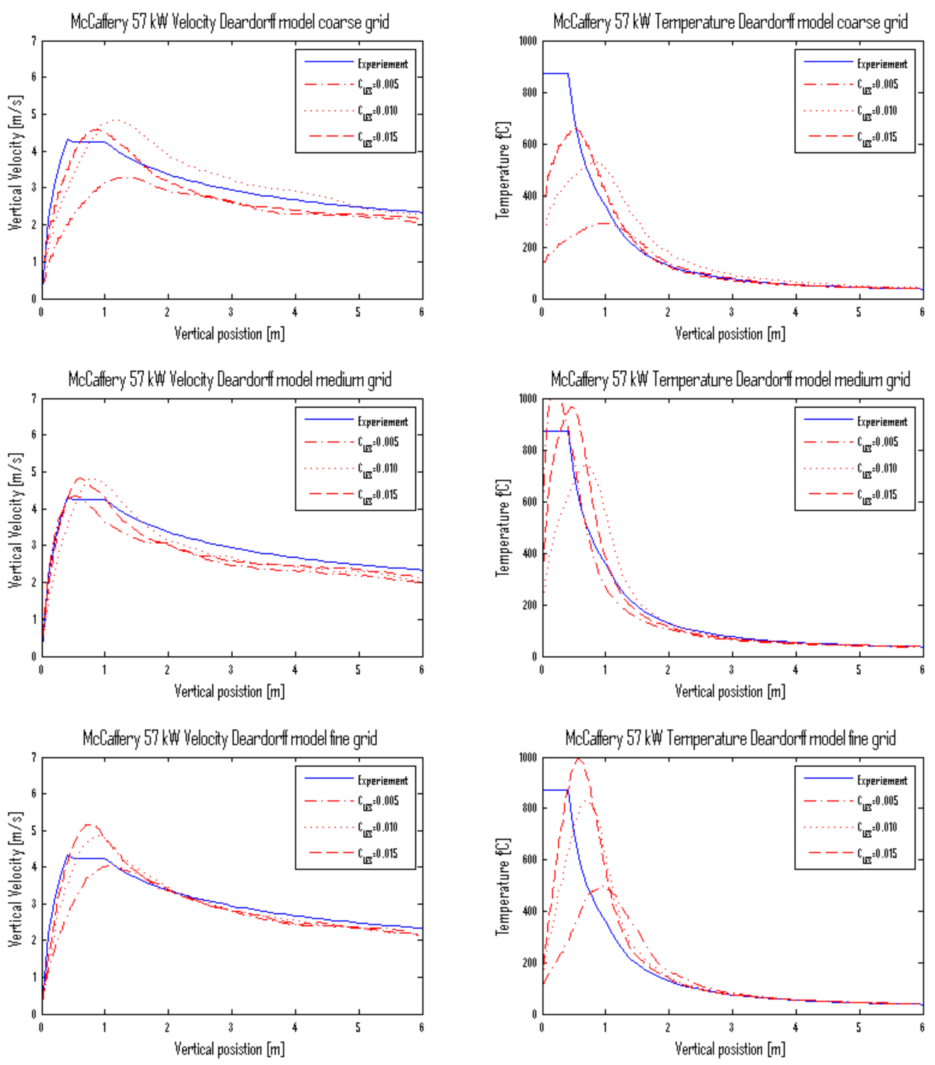


Figure 5.5: McCaffery correlation 57 kW, centerline velocity (left side) and temperature (right side) profiles with Deardorff turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

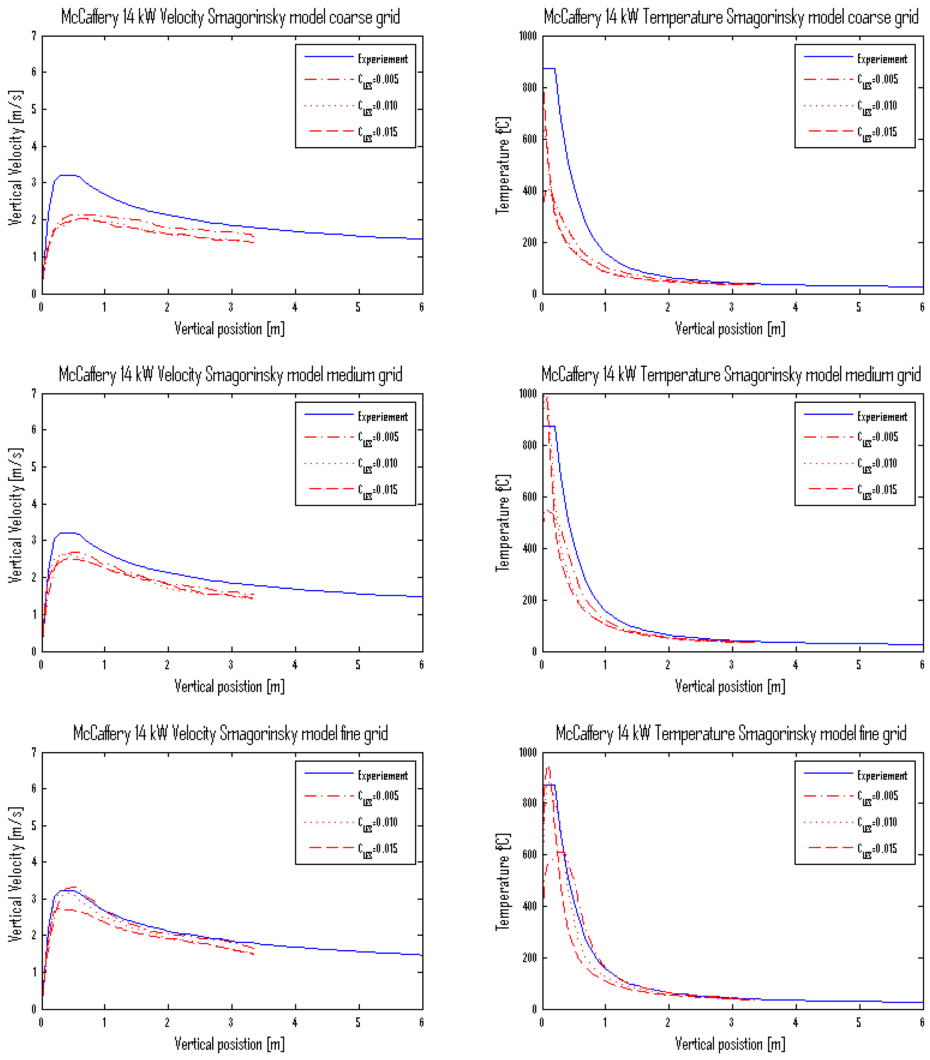


Figure 5.6: McCaffery correlation 14 kW, centerline velocity (left side) and temperature (right side) profiles with Smagorinsky turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

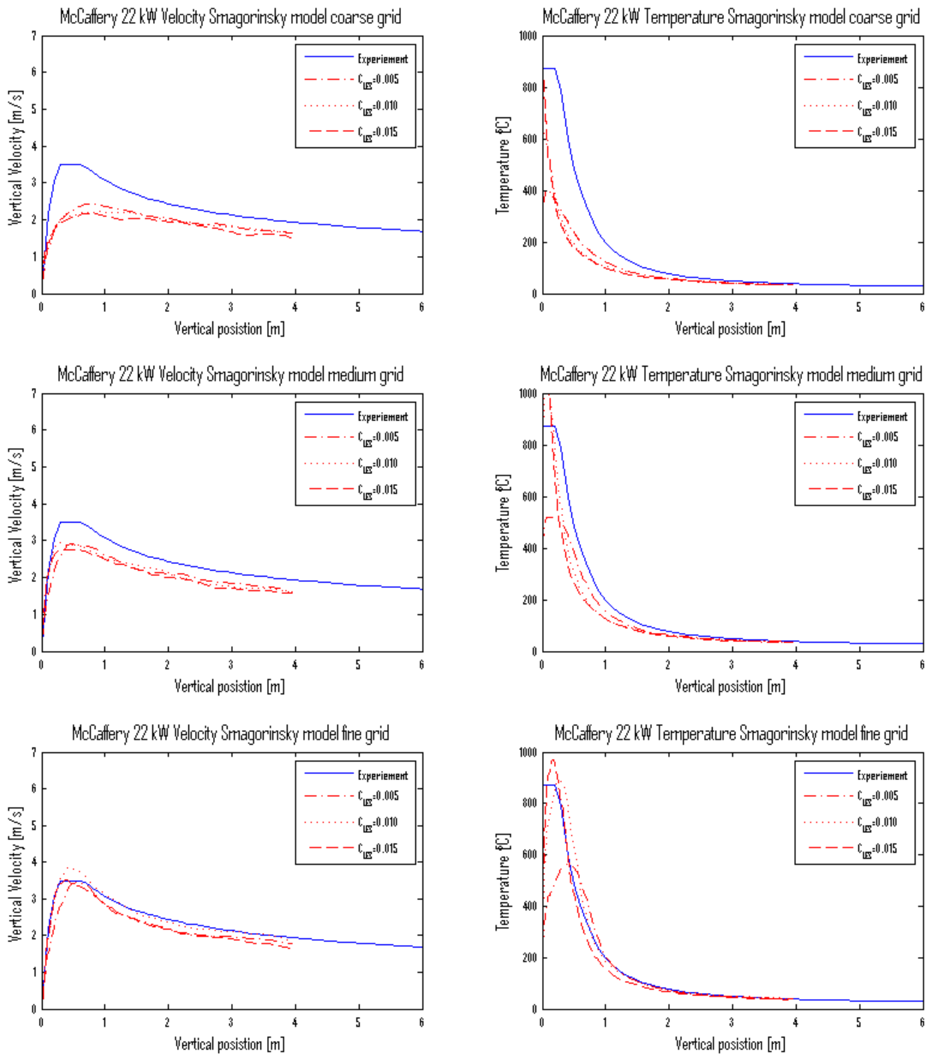


Figure 5.7: McCaffery correlation 22 kW, centerline velocity (left side) and temperature (right side) profiles with Smagorinsky turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

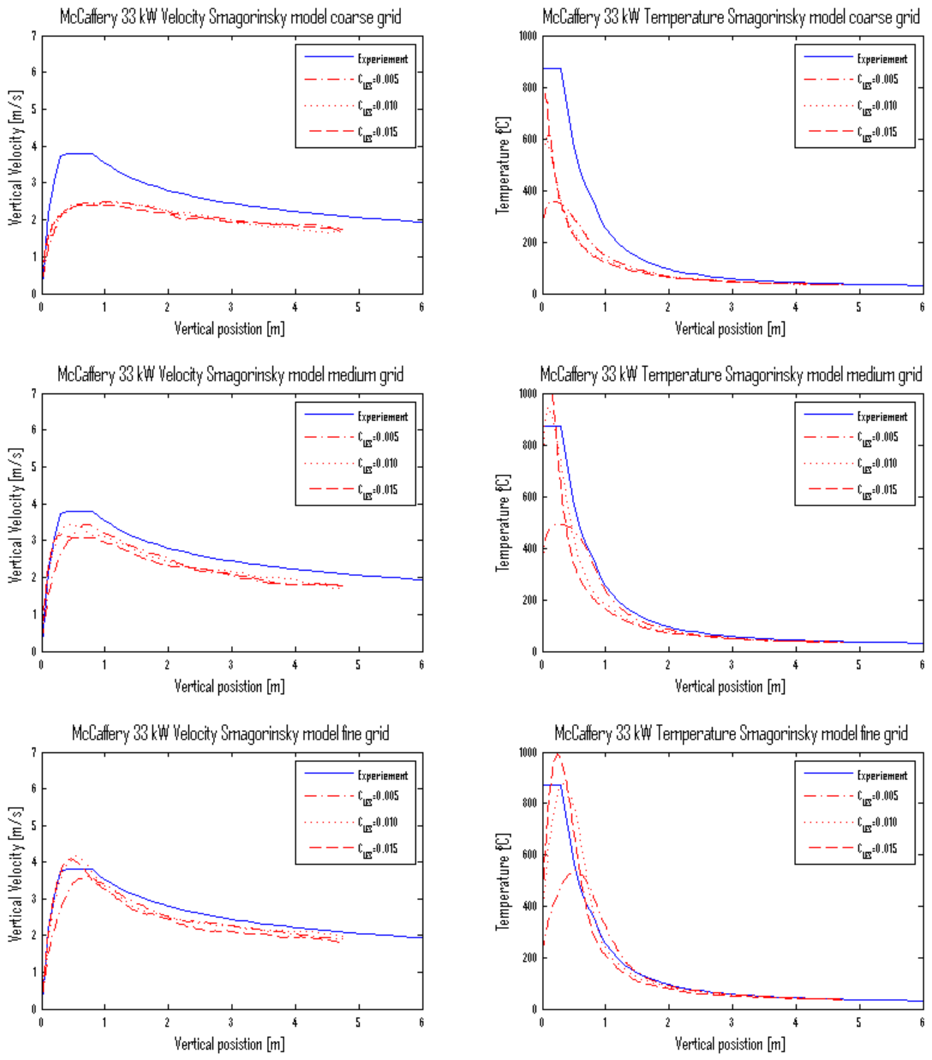


Figure 5.8: McCaffery correlation 33 kW, centerline velocity (left side) and temperature (right side) profiles with Smagorinsky turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

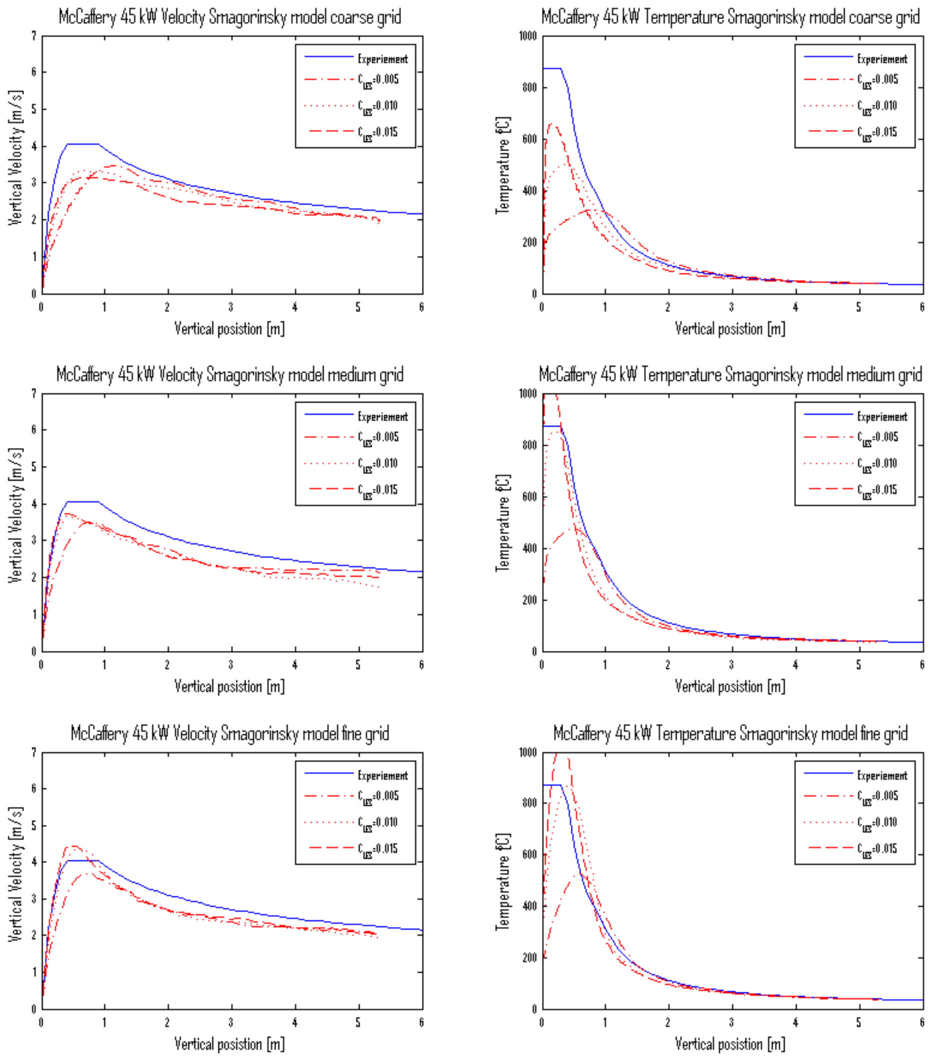


Figure 5.9: McCaffery correlation 45 kW, centerline velocity (left side) and temperature (right side) profiles with Smagorinsky turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

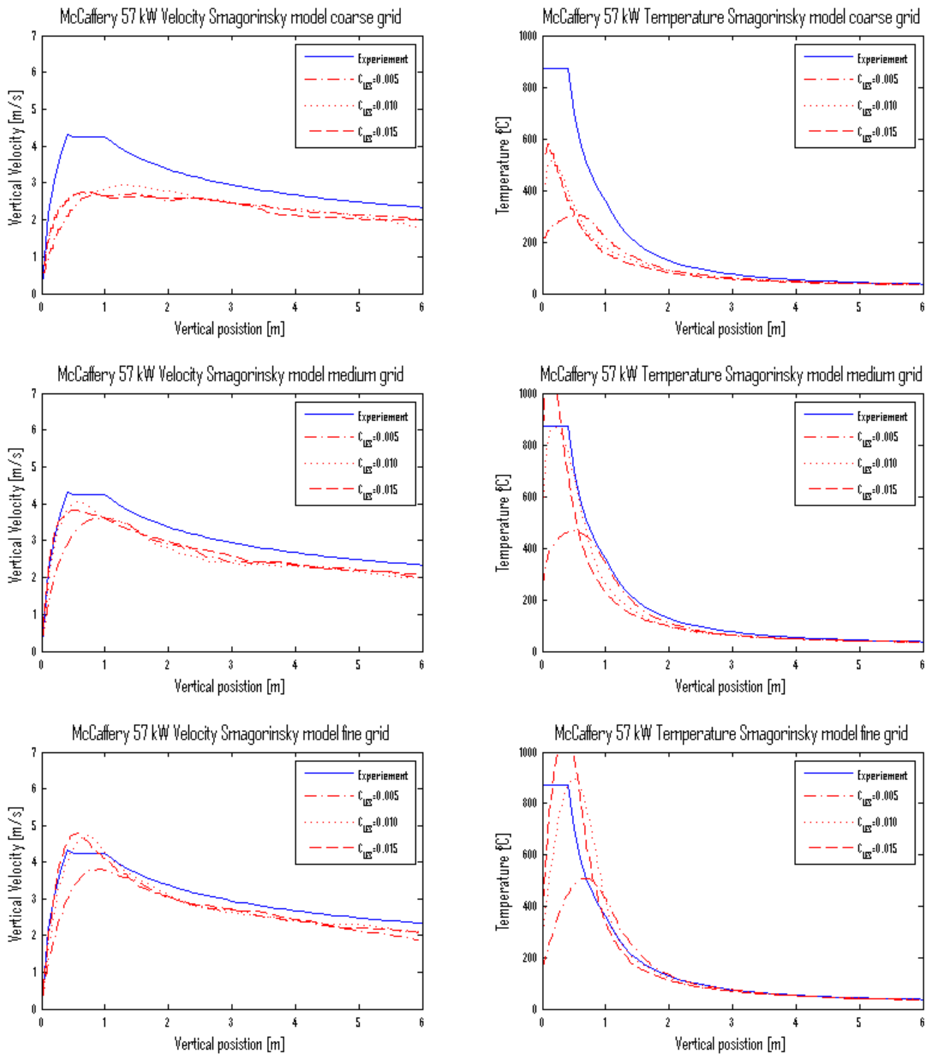


Figure 5.10: McCaffery correlation 57 kW, centerline velocity (left side) and temperature (right side) profiles with Smagorinsky turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

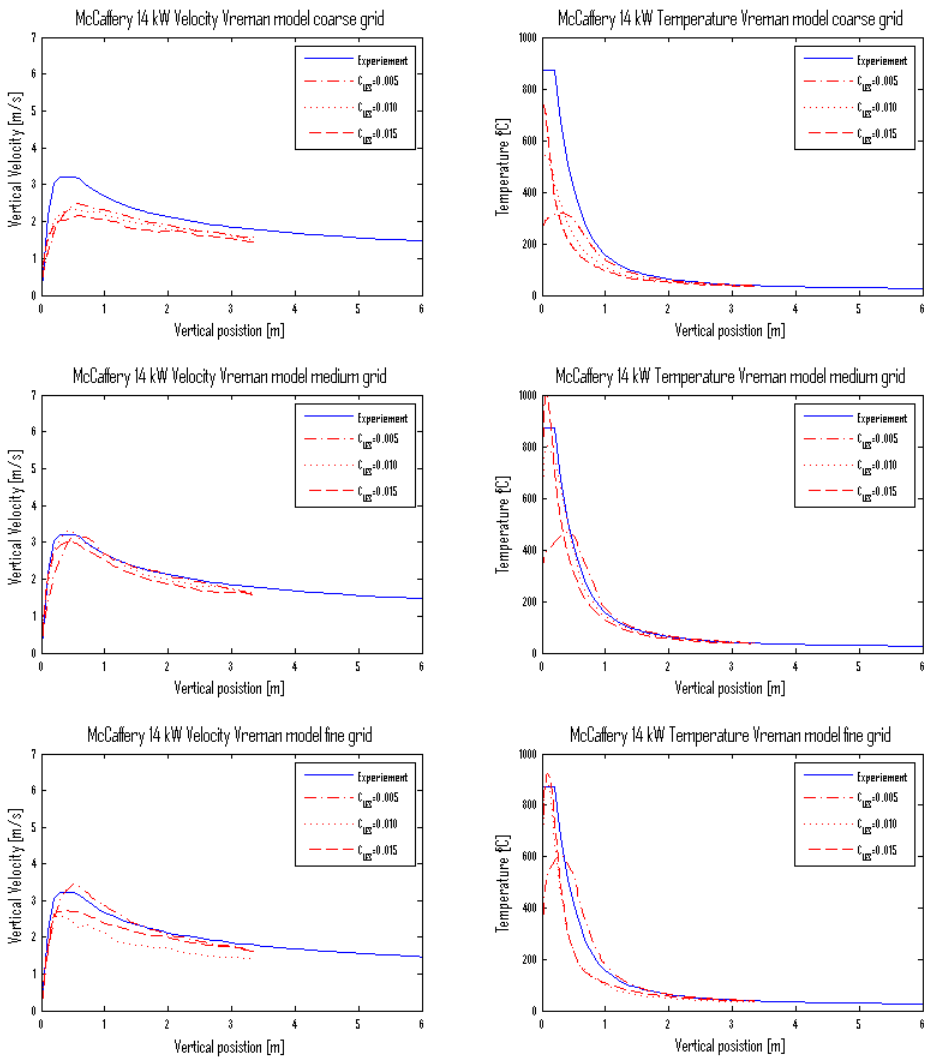


Figure 5.11: McCaffery correlation 14 kW, centerline velocity (left side) and temperature (right side) profiles with Vreman turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

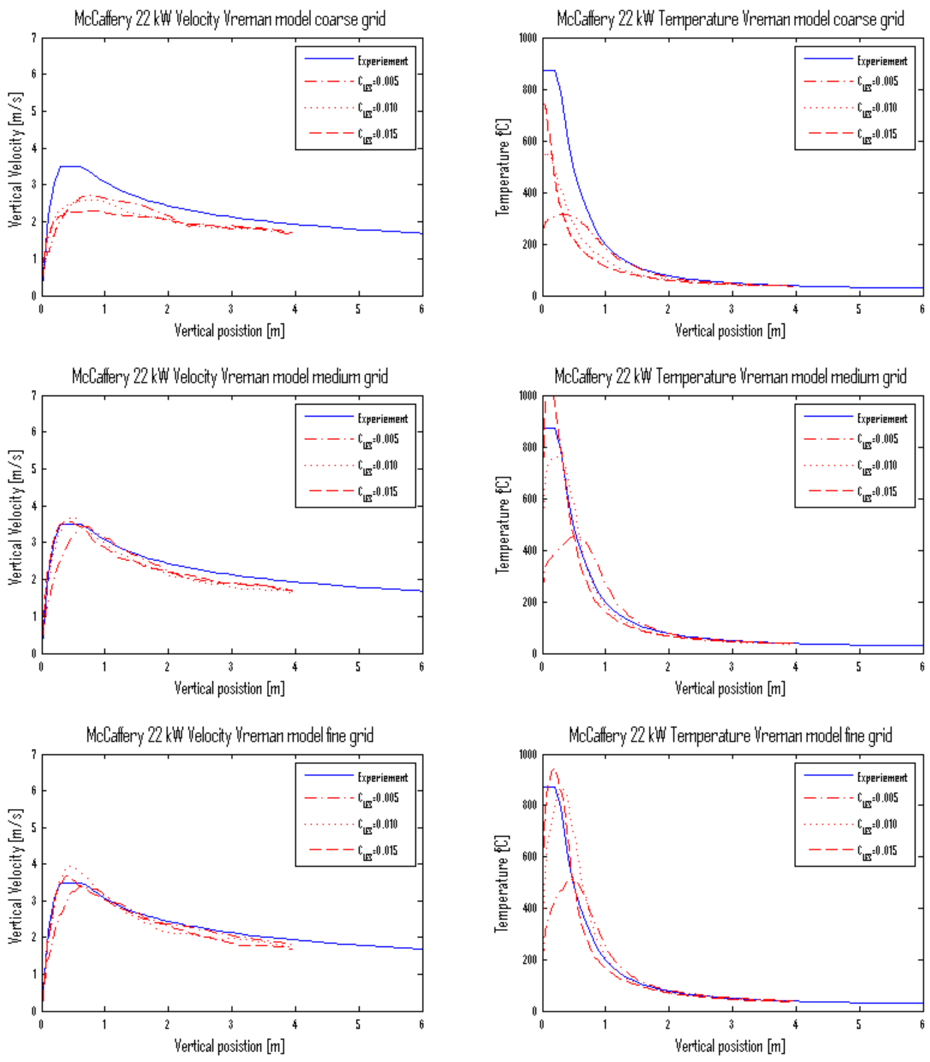


Figure 5.12: McCaffery correlation 22 kW, centerline velocity (left side) and temperature (right side) profiles with Vreman turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

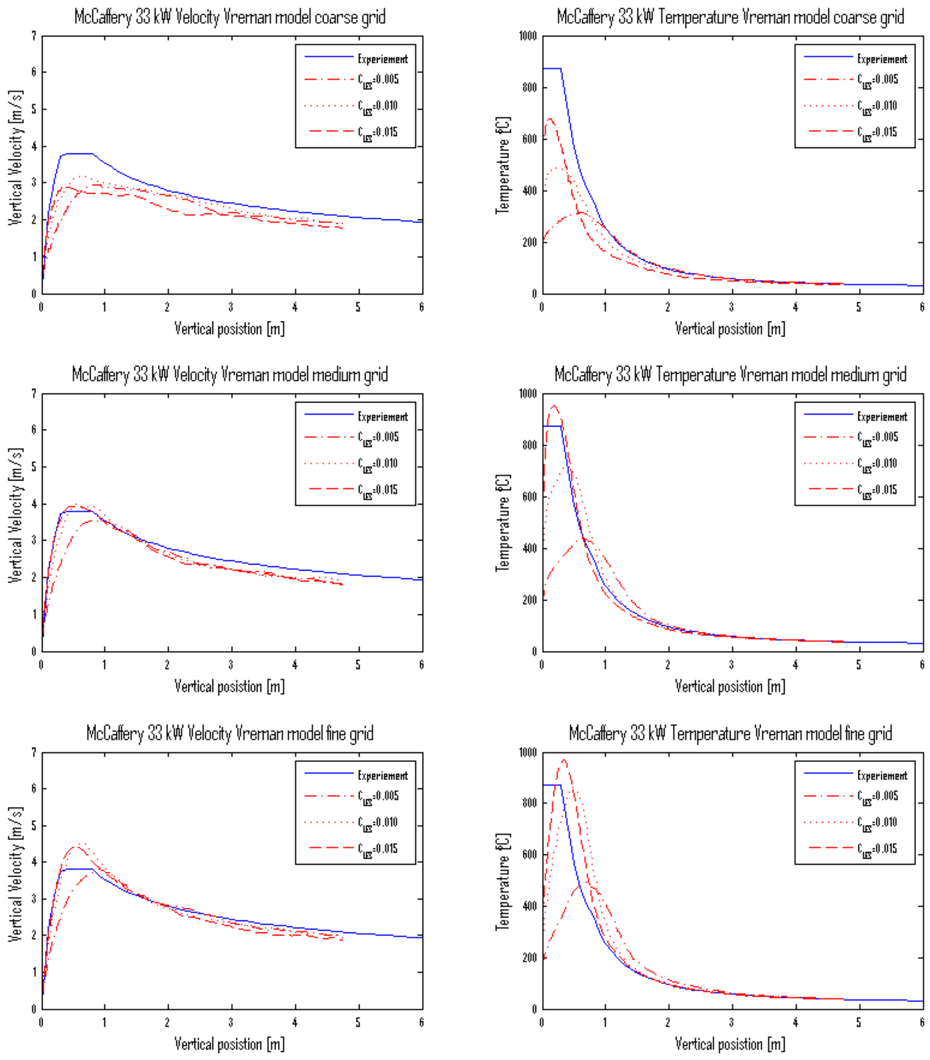


Figure 5.13: McCaffery correlation 33 kW, centerline velocity (left side) and temperature (right side) profiles with Vreman turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

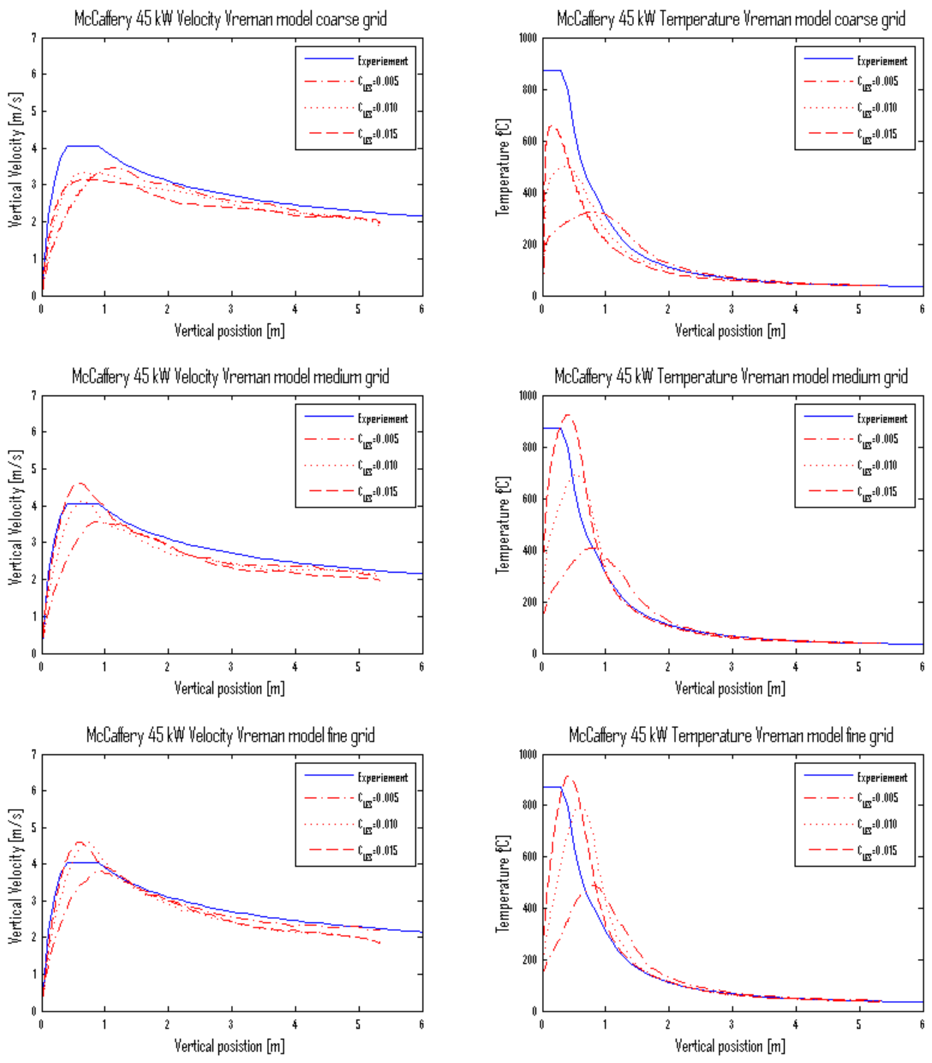


Figure 5.14: McCaffery correlation 45 kW, centerline velocity (left side) and temperature (right side) profiles with Vreman turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

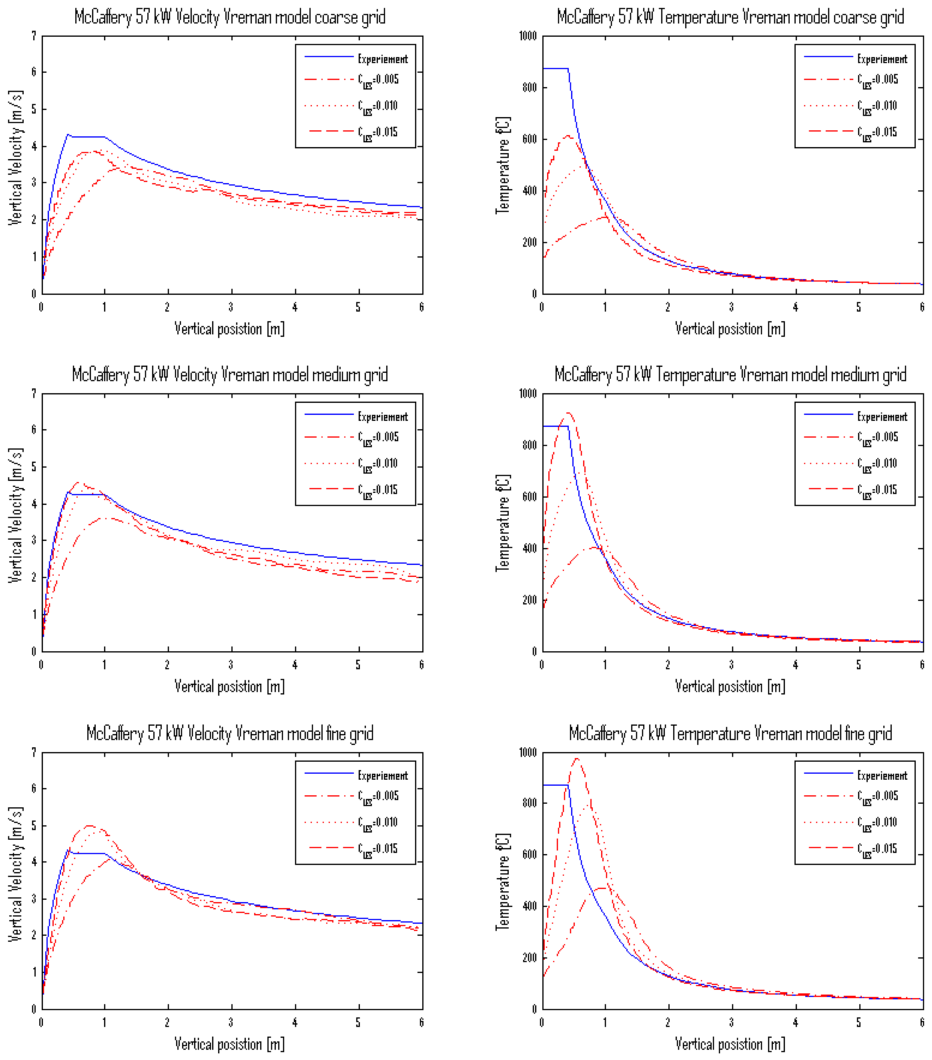


Figure 5.15: McCaffery correlation 57 kW, centerline velocity (left side) and temperature (right side) profiles with Vreman turbulence model. Coarse grid at the top, medium grid in the middle and fine grid at the bottom.

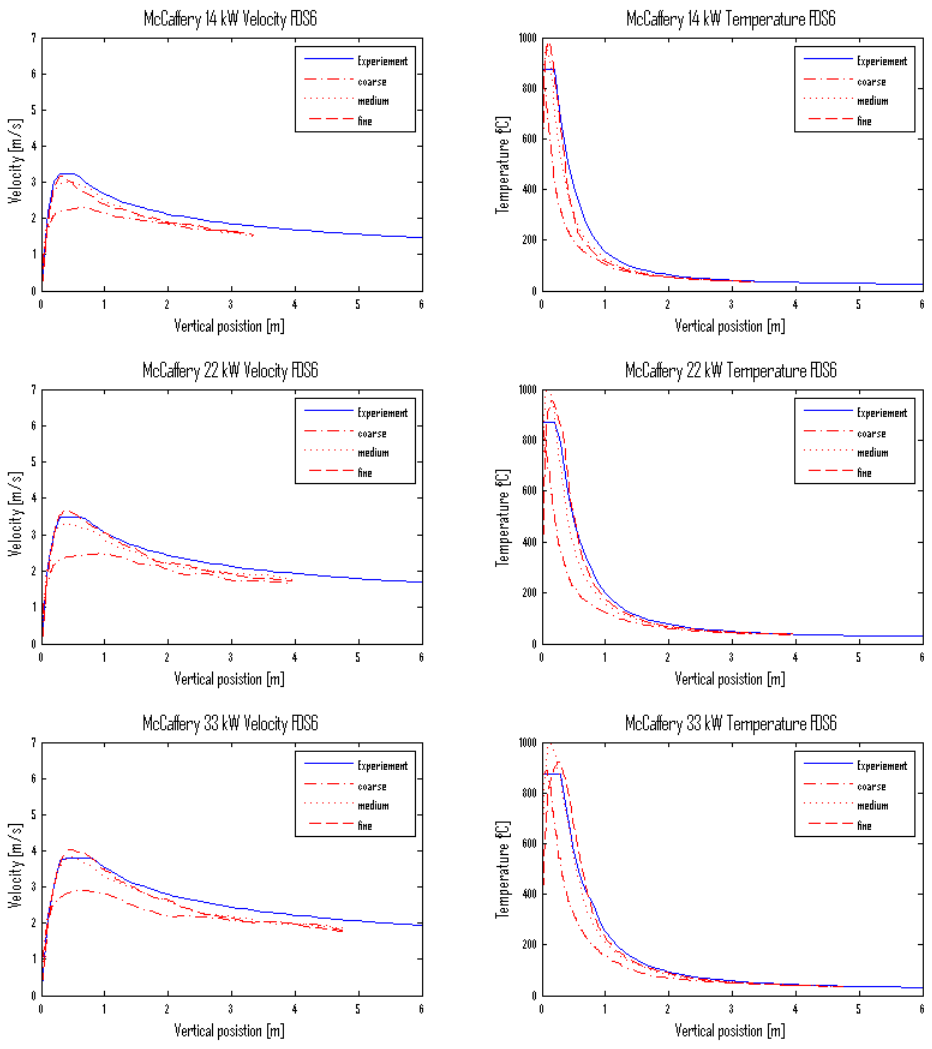


Figure 5.16: McCaffery correlation, centerline velocity (left side) and temperature (right side) profiles with FDS6 Deardorff turbulence model.

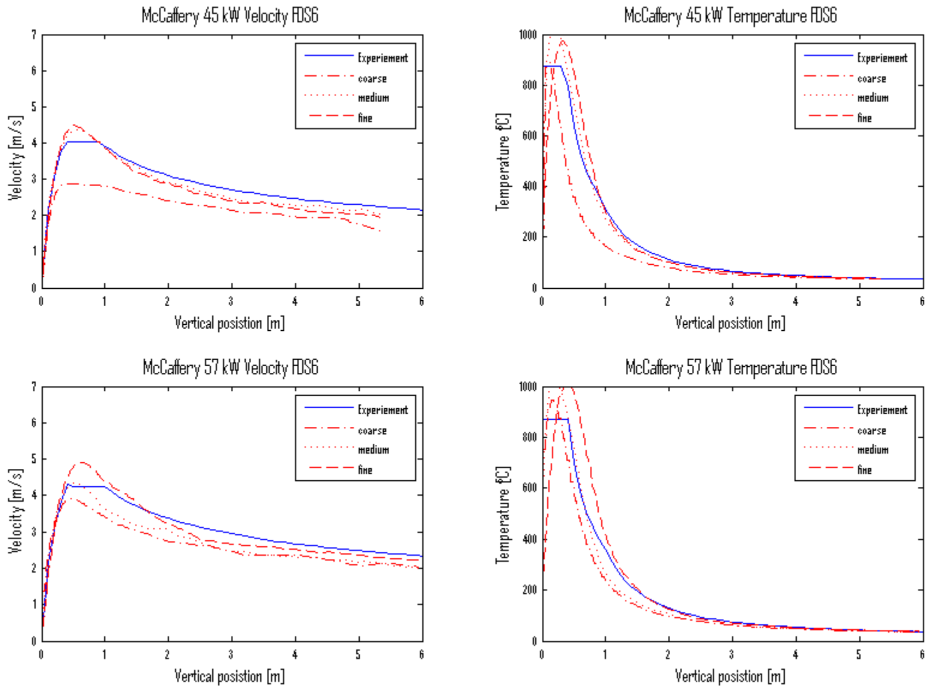


Figure 5.17: McCaffery correlation, centerline velocity (left side) and temperature (right side) profiles with FDS6 Deardorff turbulence model.

5.2 Heskestad Flame Height Correlation

The different fire cases for Heskestad validation is found in Table 5.1. Propane is used as fuel with $0.1 \leq Q^* \leq 10000$ at atmospheric pressure and 20 °C. Three different grids; 17 x 17 x 40, 33 x 33 x 80 and 65 x 65 x 160 and three different C_{LES}; 0.005, 0.01 and 0.015 are simulated for the turbulence models Deardorff, (dynamic) Smagorinsky and Vreman. D^* and HRR is presented in Table 5.1.

Table 5.1: Heskestad Flame Height Simulations [32].

Q^*	$\dot{Q}$ [kW]	D^* [-]	dx_{10} [m]
0.1	151	0.45	0.045
0.2	303	0.59	0.059
0.5	756	0.86	0.086
1	1513	1.13	0.113
2	3025	1.49	0.149
5	7564	2.15	0.215
10	15127	2.84	0.284
20	30255	3.75	0.375
50	75636	5.40	0.540
100	151273	7.13	0.713
200	302545	9.41	0.941
500	756363	13.6	1.36
1000	1512725	17.9	1.79
2000	3025450	23.6	2.36
5000	7563625	34.1	3.41
10000	15127250	45.0	4.50

In the validation of flame height by Heskestad correlation, outputs from the simulations must be post-processed to predict the flame height. A FORTRAN programme for this is available at `http://code.google.com/p/fds-smv/`. The output from the simulation is heat release per unit length ($HRRPUL = \int \dot{q}'''dxdy$). The flame height, L_f, is assumed to be where 99 % of fuel is consumed on average. This is the same as the height where $\sum_0^{L_f} HRRPUL = 0.99 \cdot \sum_0^{\infty} HRRPUL$. In line 107 in Appendix C the flame height is found through interpolation.

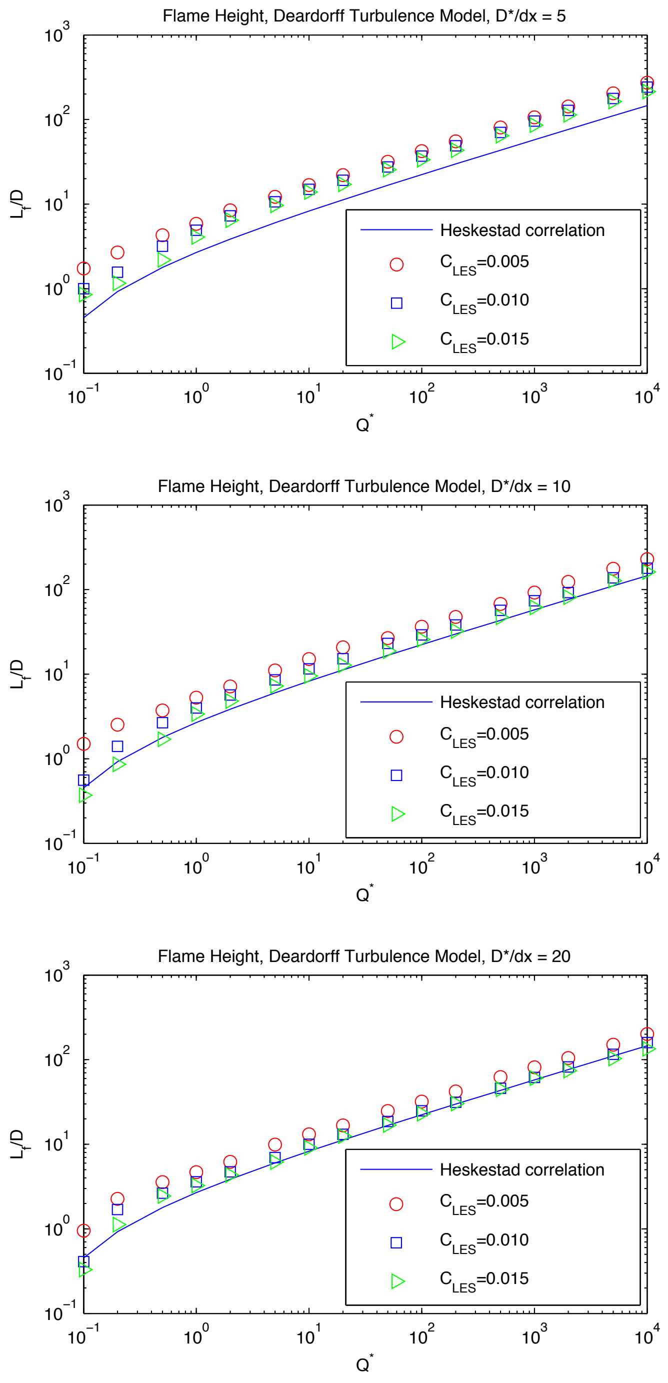


Figure 5.18: Heskestad Flame Height Correlation Deardorff Turbulence Model.

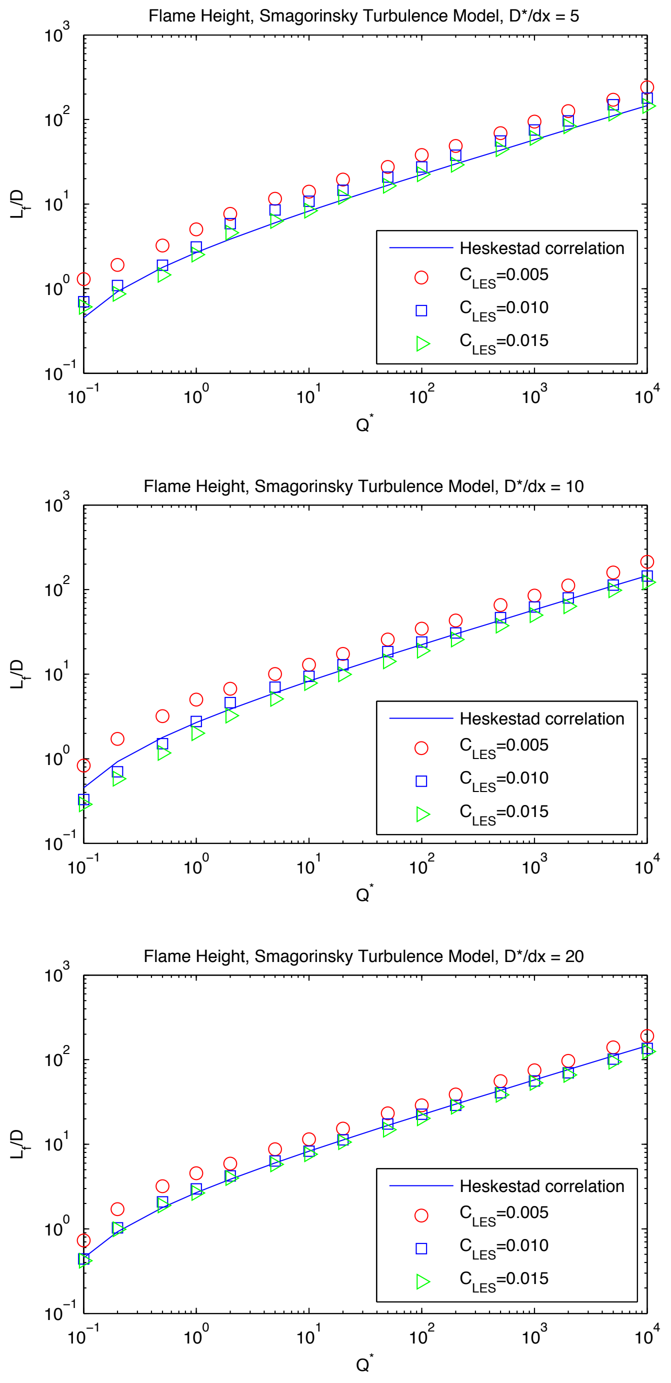


Figure 5.19: Heskestad Flame Height Correlation Smagorinsky Turbulence Model.

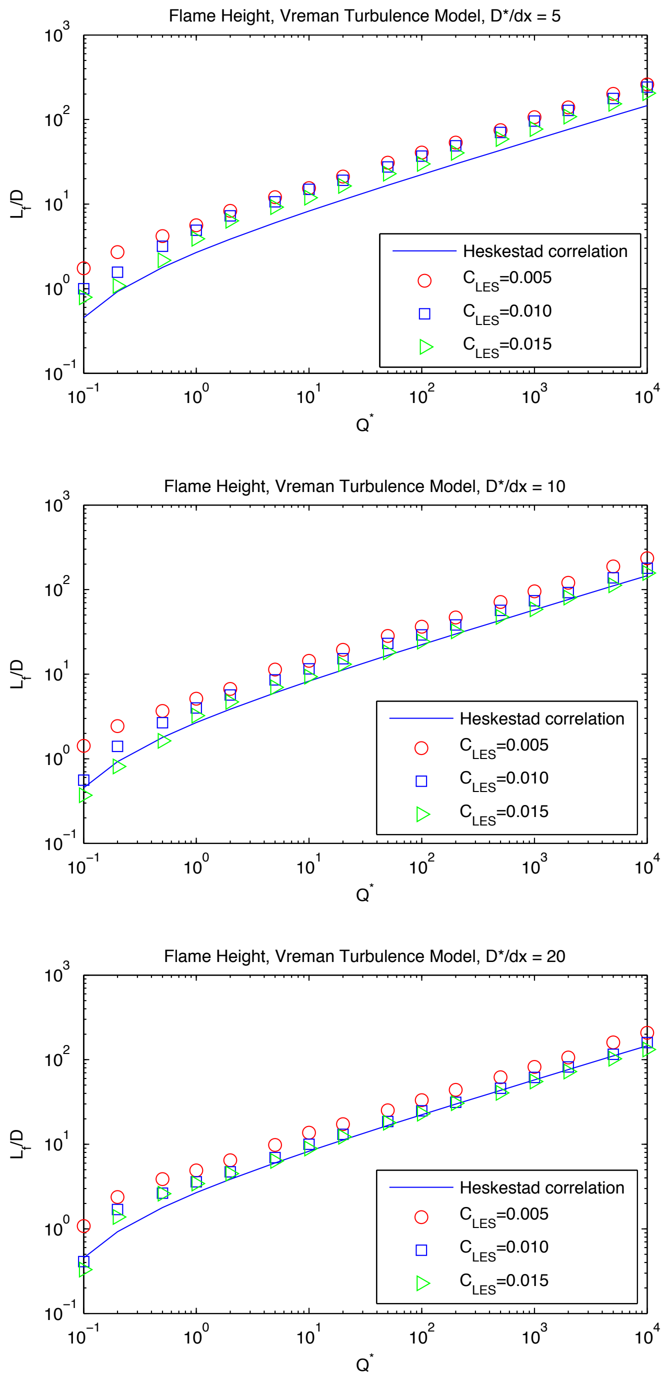
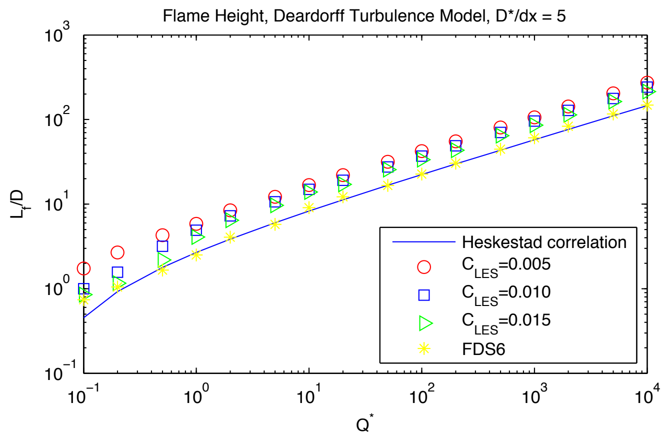
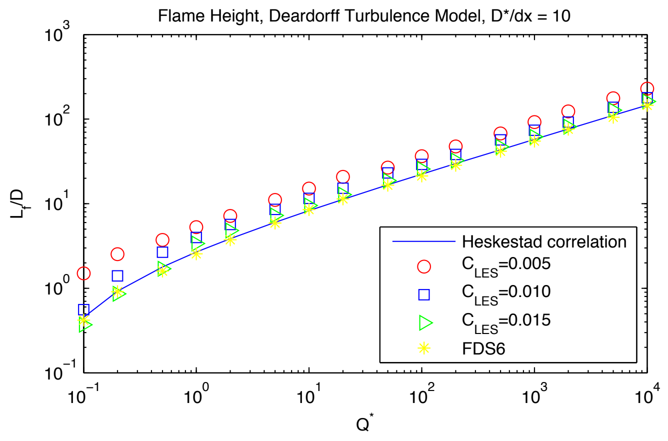


Figure 5.20: Heskestad Flame Height Correlation, Vreman Turbulence Model.





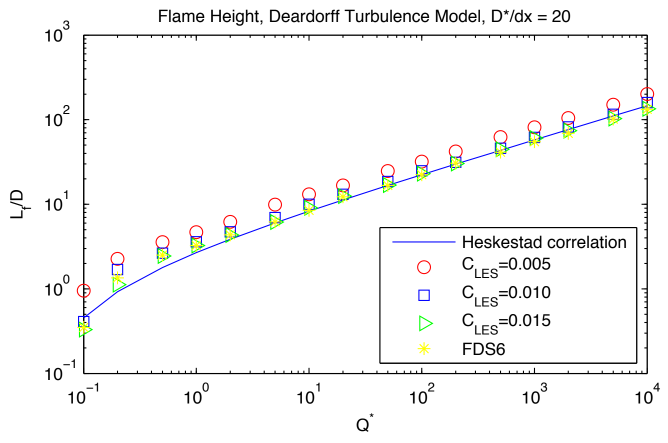


Figure 5.21: Heskestad Flame Height Correlation FDS6 compared with FDS-EDC, Deardorff Turbulence Model.

5.3 Sandia Plume

Sandia Plume experiments are specifically designed for validating CFD models involving fire plumes. The Fire Laboratory for Accreditation of Models by Experimentation (FLAME) facility is located in New Mexico and is where the experiments are performed by Tieszen *et al.* [33]. The experiments are arranged with a 0.5 m steel plane surrounding the fire area of 1 m in diameter. Average velocities profiles are measured in heights of 0.3 m, 0.5 m and 0.9 m above the fire (see Figure 5.59) with Planar Laser Induced Fluorescence (PLIF).

Simulations is compared with the average velocity profiles between 10-20 seconds in the simulations. The computational domain size is 3 m x 3 m x 4 m meters divided in uniform rectangular grid cell of 1.5 cm, 3.0 cm and 6.0 cm. This corresponds to grid resolutions of 192 x 192 x 256, 96 x 96 x 128 and 48 x 48 x 64 respectively. The fuel releases were 0.04 kg/m^{2}s (test 14), 0.053 kg/m^{2}s (test 24) and 0.066 kg/m^{2}s (test 17). Dimensionless HRR and fire diameter is given in Table 5.2. Input for all cases are specified after the actual test conditions. Example of inputs for the fires are seen in Appendix A.1-A.3 [33].

Table 5.2: Sandia plume experiment simulations.

Testnumber	$\dot{Q}$	Q^*	D^*
	[kW]	[-]	[-]
14	1590	1.35	1.13
17	2610	2.22	1.38
24	2070	1.76	1.25

5.3.1 Methane Fire, Test 14

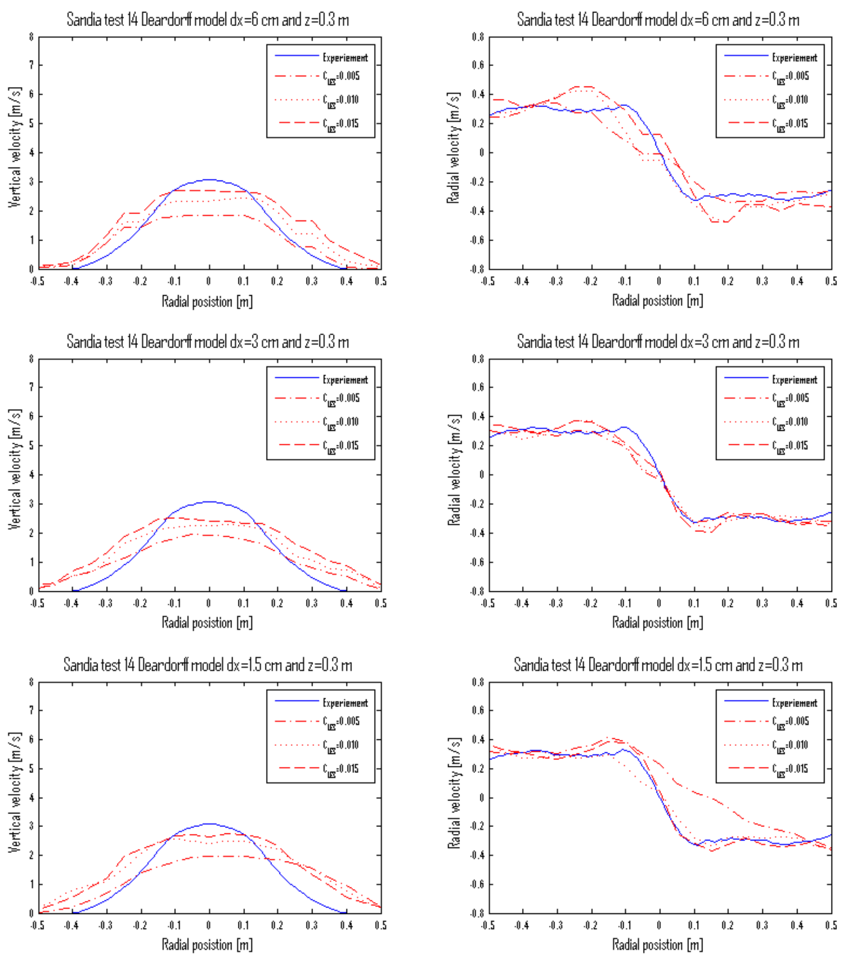


Figure 5.22: Sandia plume experiment test 14 with Deardorff turbulence model at z = 0.3 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

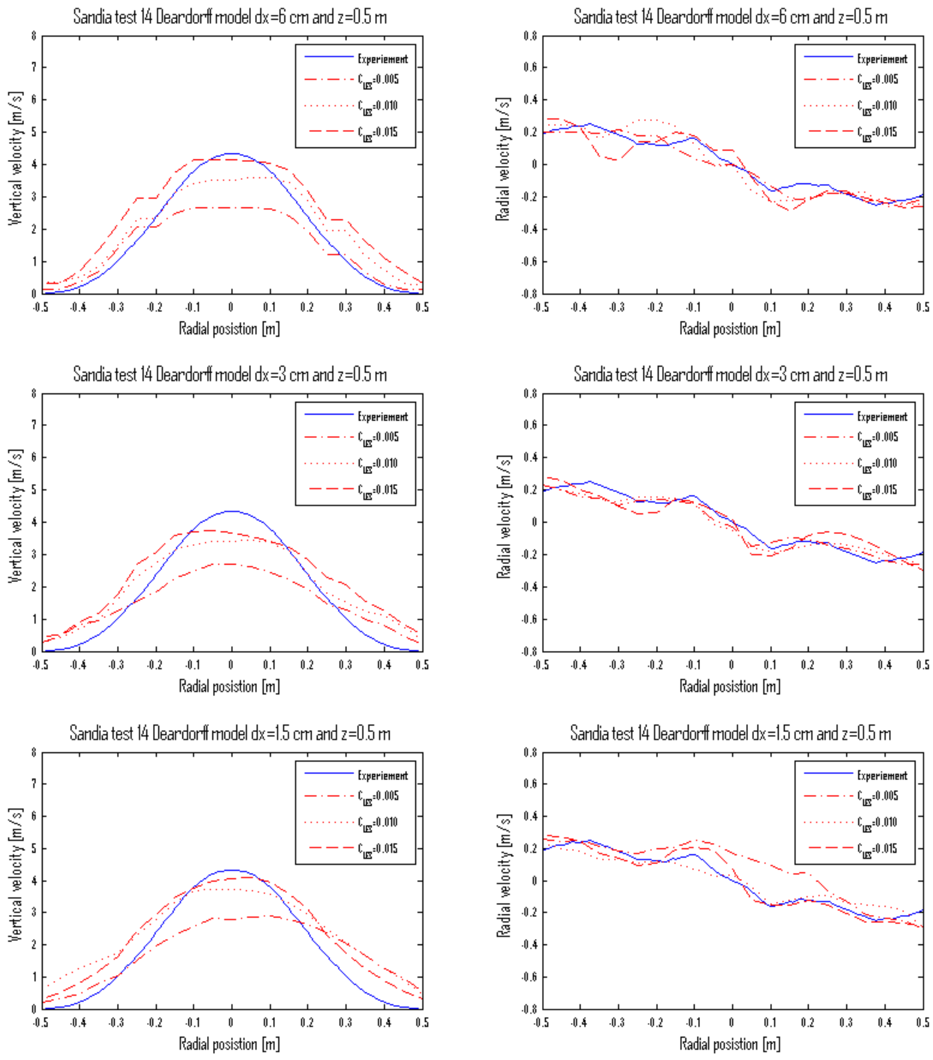


Figure 5.23: Sandia plume experiment test 14 with Deardorff turbulence model at z = 0.5 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

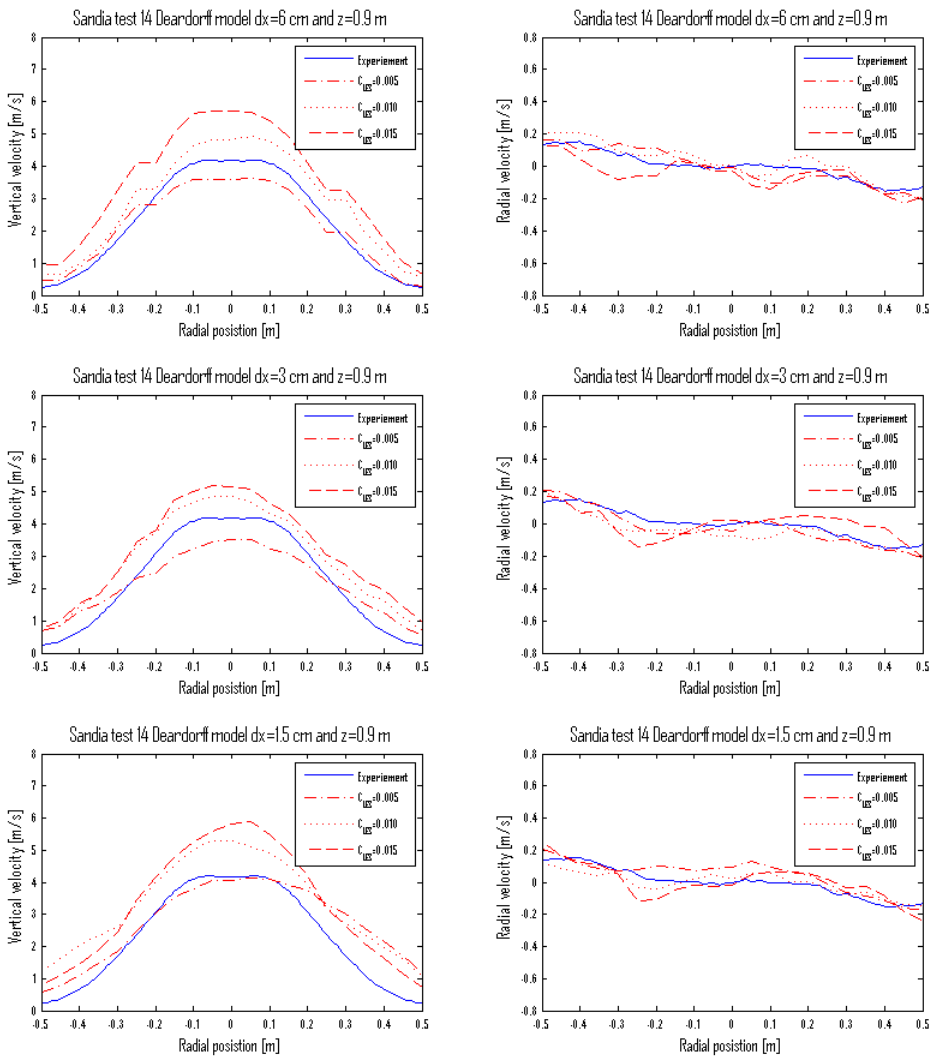


Figure 5.24: Sandia plume experiment test 14 with Deardorff turbulence model at z = 0.9 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

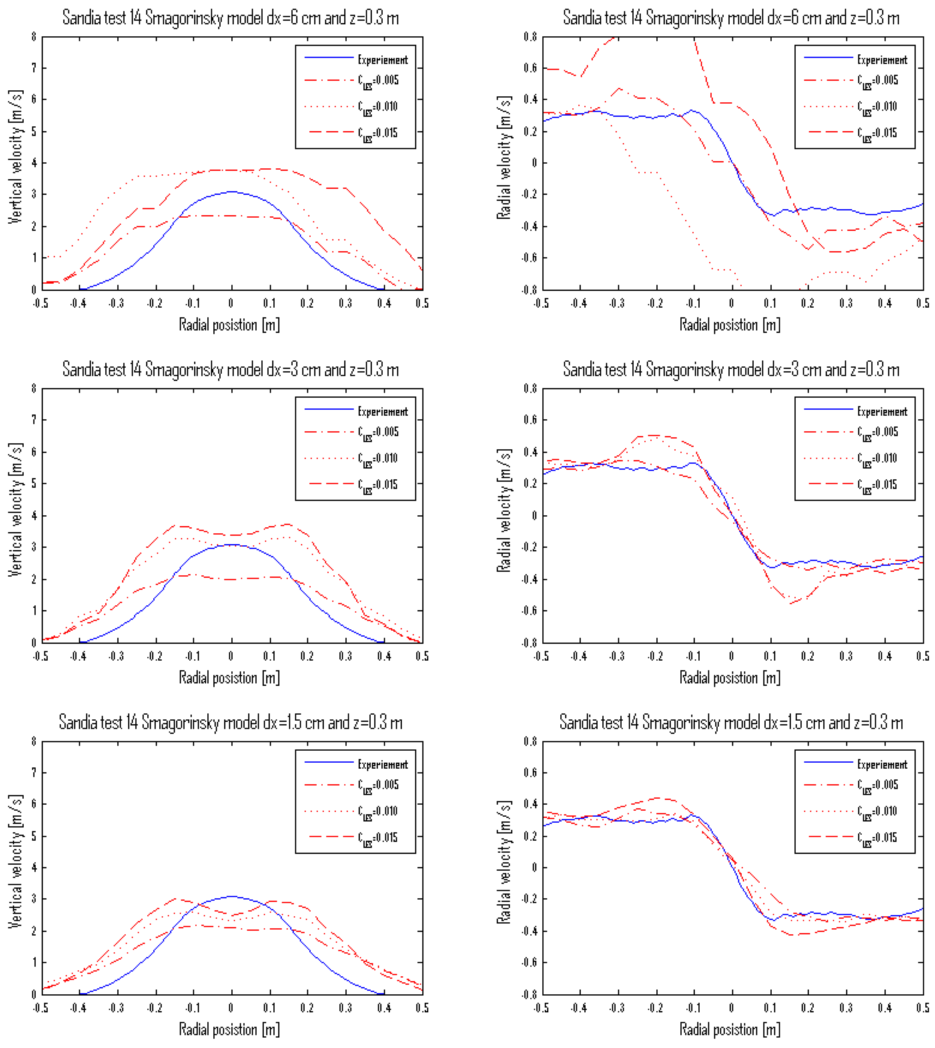


Figure 5.25: Sandia plume experiment test 14 with Smagosinky turbulence model at z = 0.3 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

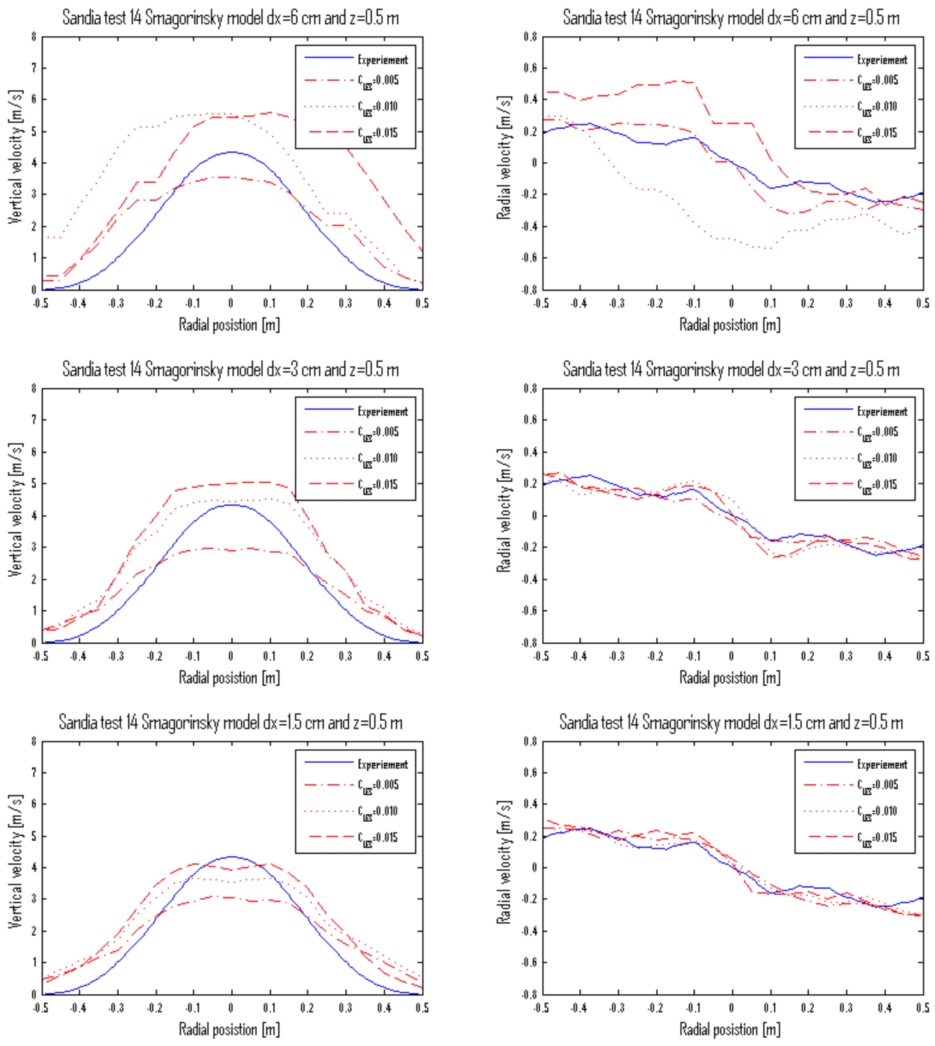


Figure 5.26: Sandia plume experiment test 14 with Smagosinky turbulence model at z = 0.5 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

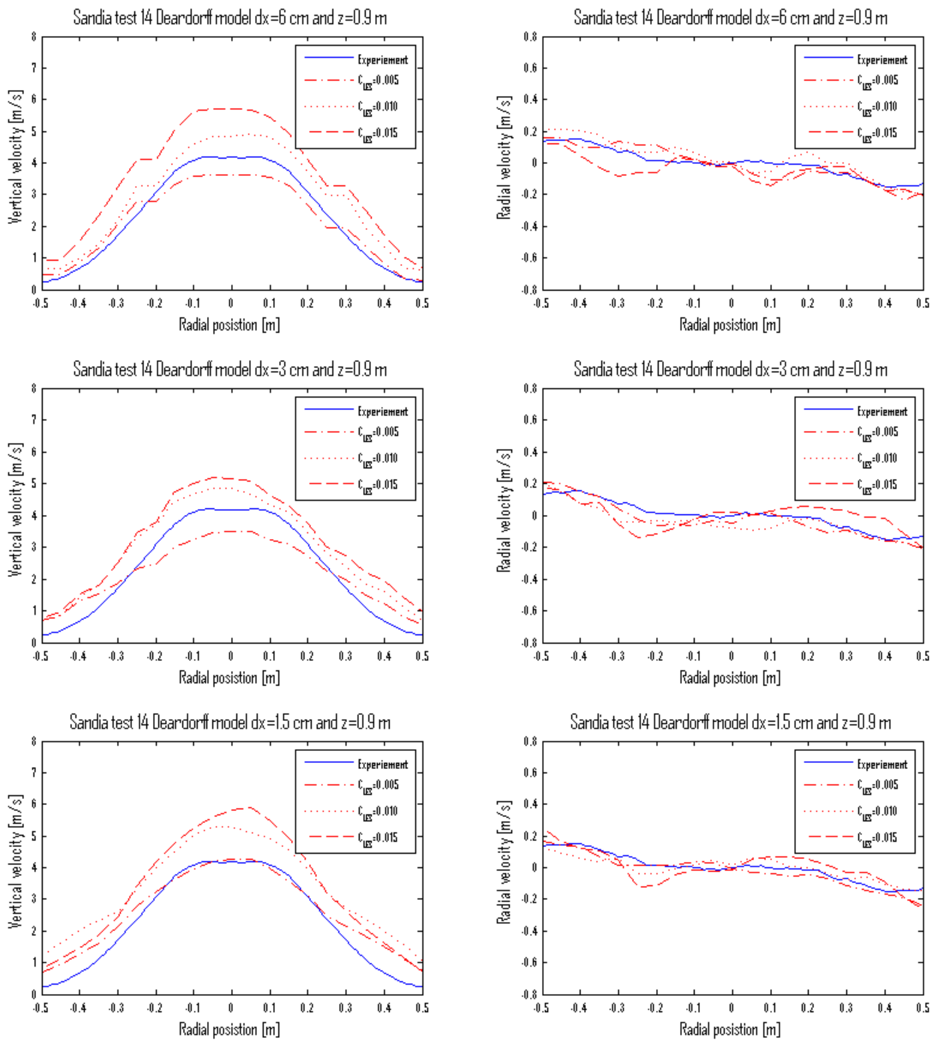


Figure 5.27: Sandia plume experiment test 14 with Smagosinky turbulence model at z = 0.9 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

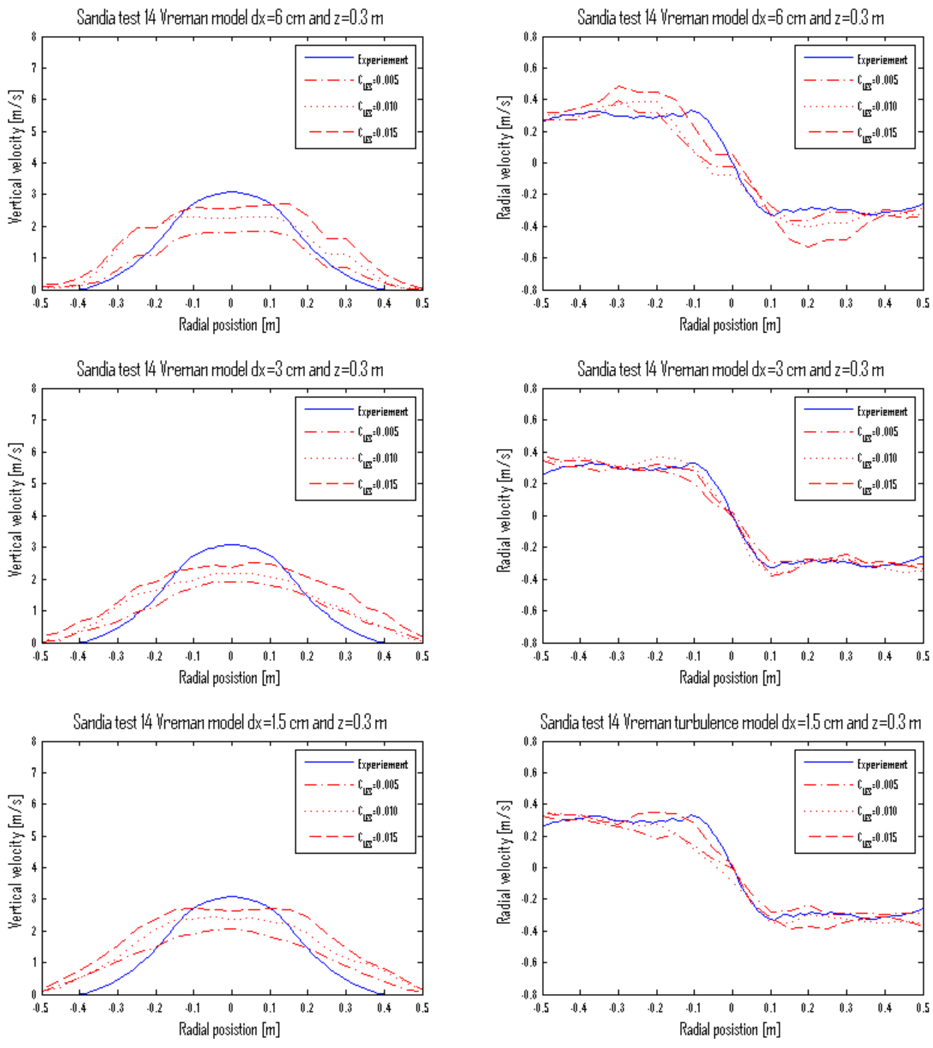


Figure 5.28: Sandia plume experiment test 14 with Vreman turbulence model at z = 0.3 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

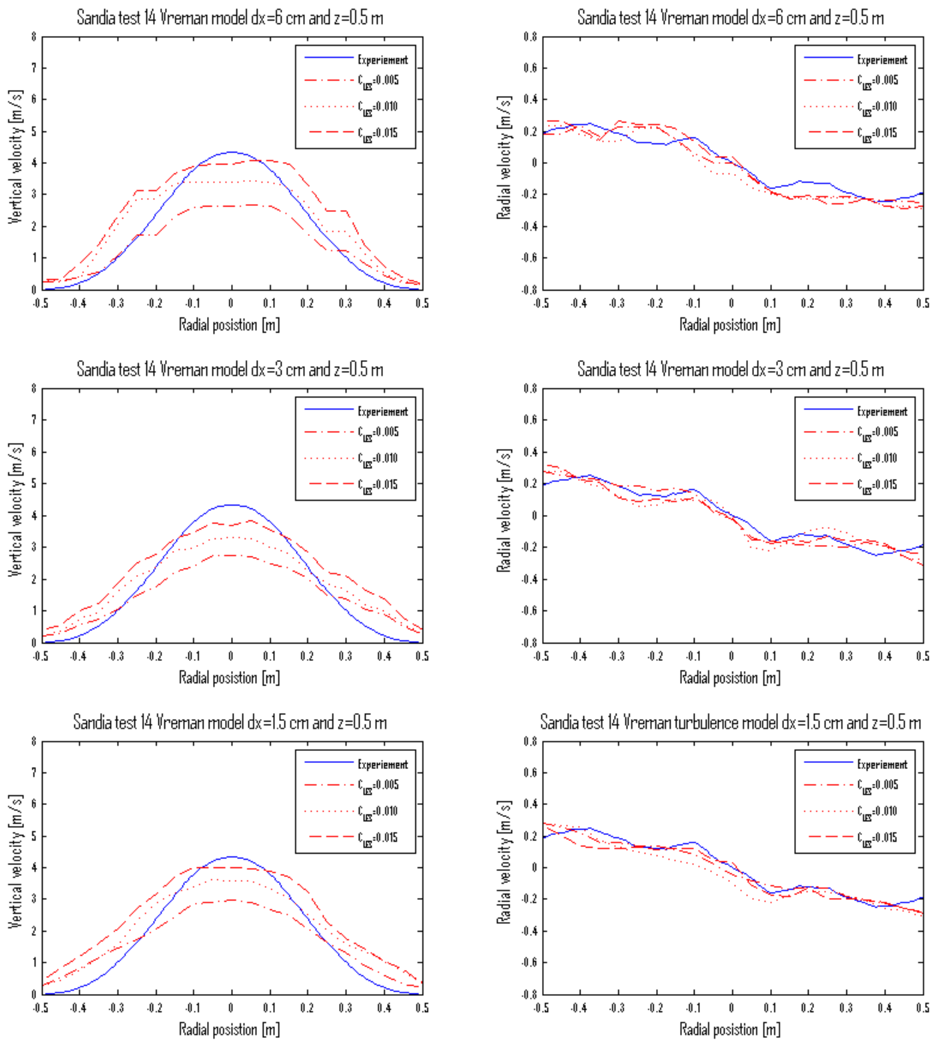


Figure 5.29: Sandia plume experiment test 14 with Vreman turbulence model at z = 0.5 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

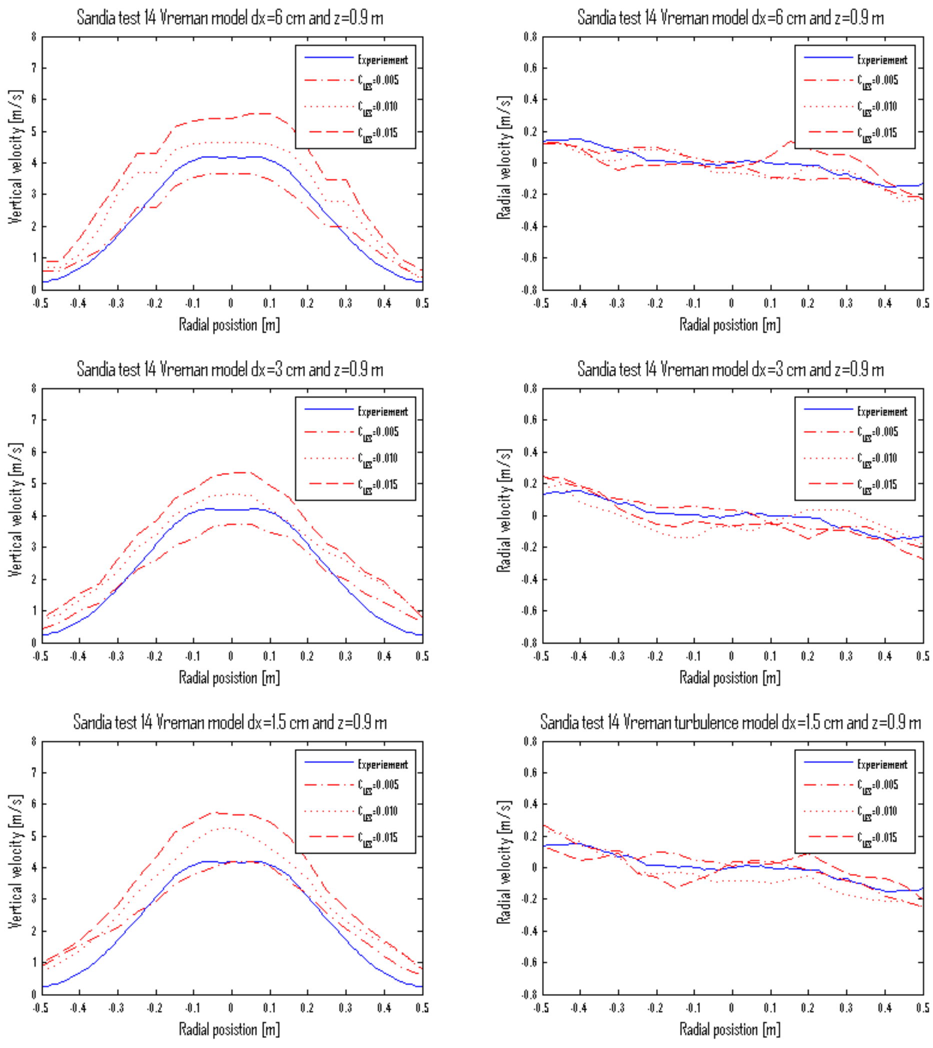


Figure 5.30: Sandia plume experiment test 14 with Vreman turbulence model at z = 0.9 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

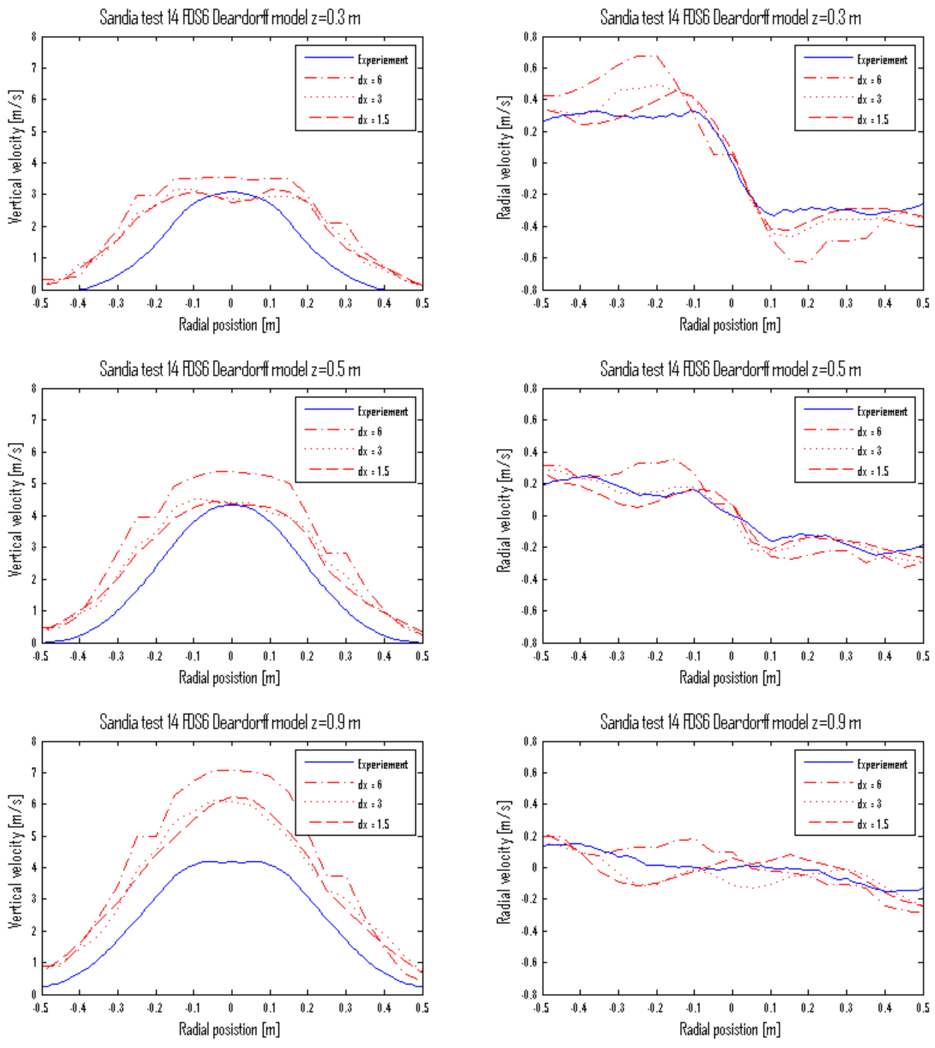


Figure 5.31: Sandia plume experiment test 14 with FDS6. Vertical velocity to the left and radial velocity to the right. On the top at z = 0.3 m, z = 0.5 m in the middle and z = 0.9 m at the bottom.

5.3.2 Methane Fire, Test 17

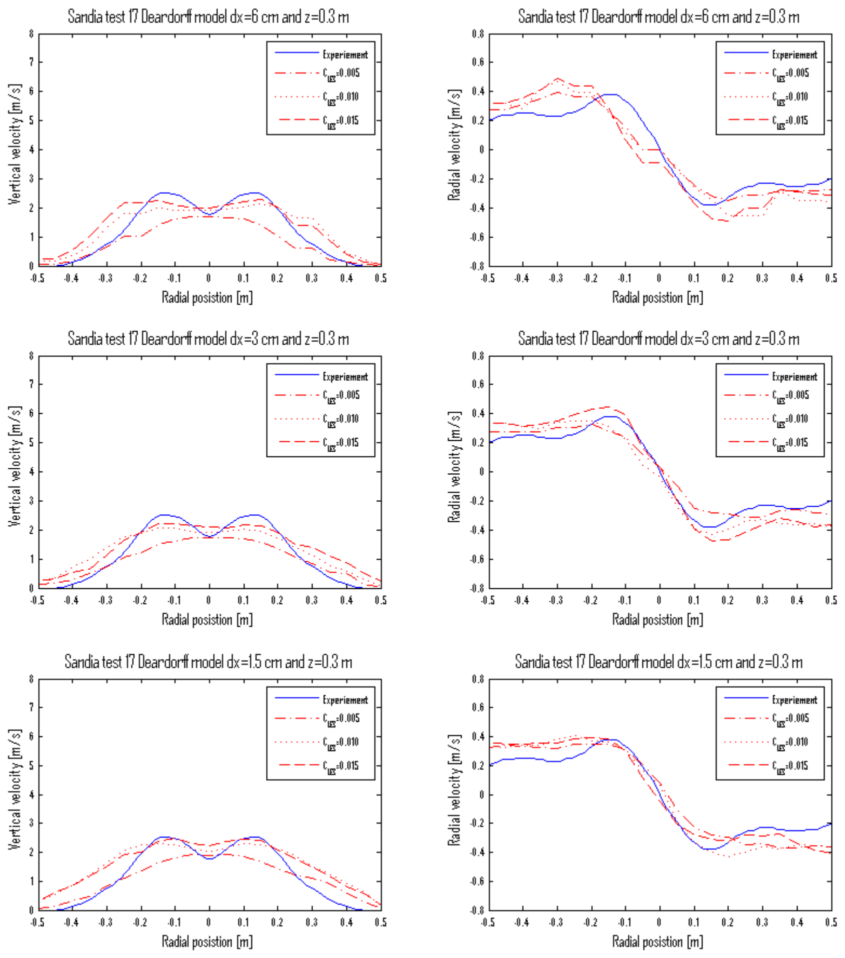


Figure 5.32: Sandia plume experiment test 17 with Deardorff turbulence model at z = 0.3 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

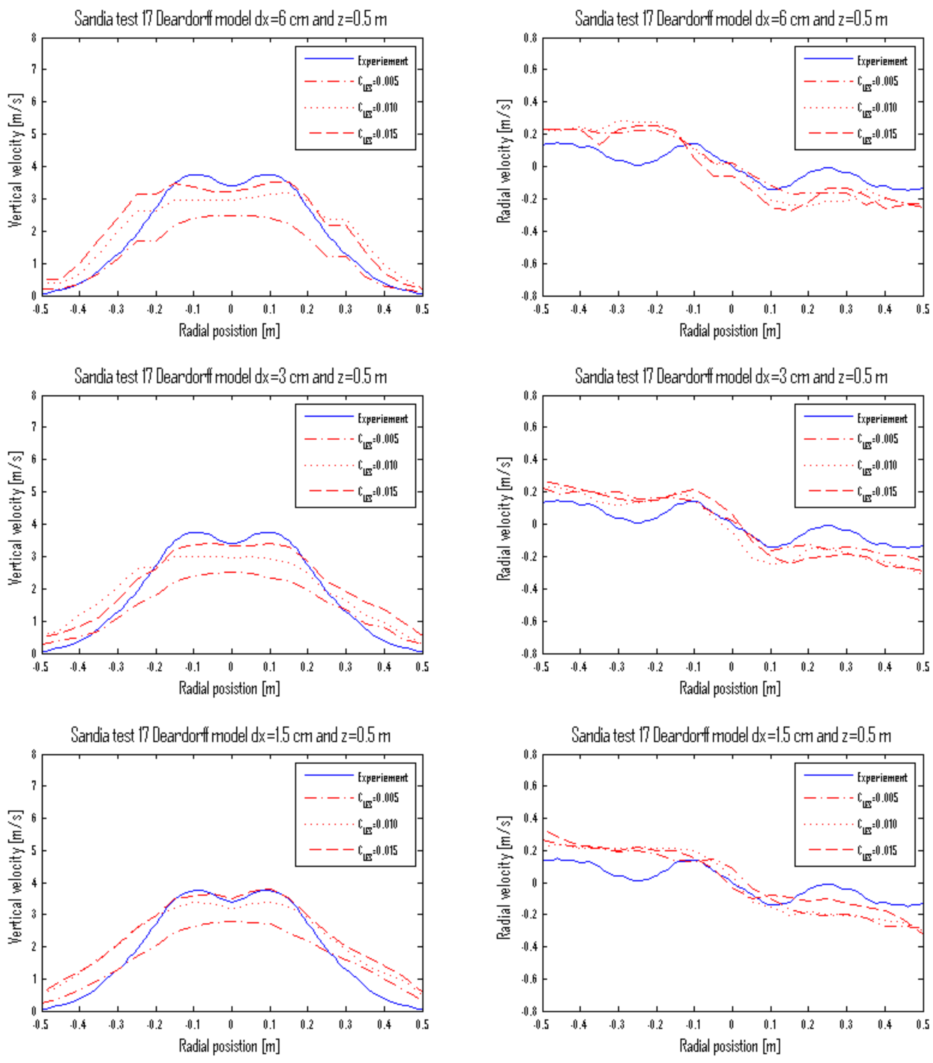


Figure 5.33: Sandia plume experiment test 17 with Deardorff turbulence model at z = 0.5 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

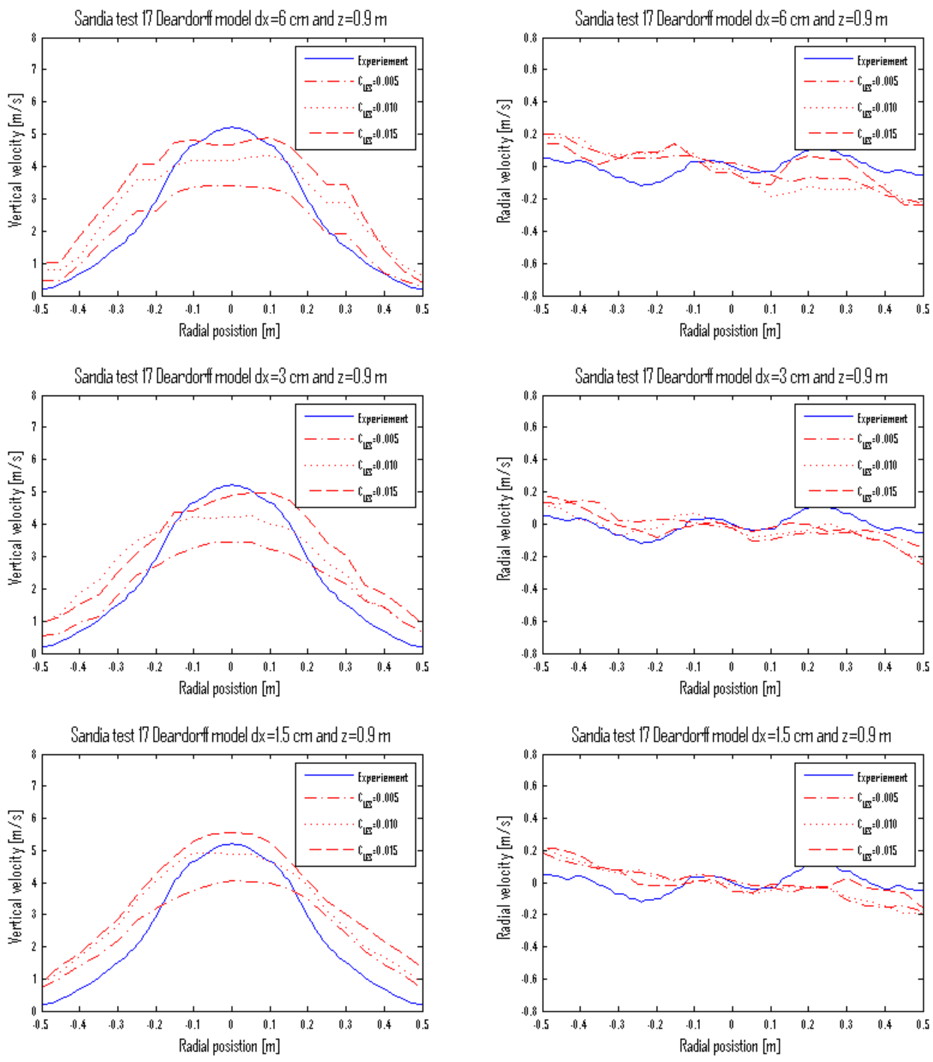


Figure 5.34: Sandia plume experiment test 17 with Deardorff turbulence model at z = 0.9 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

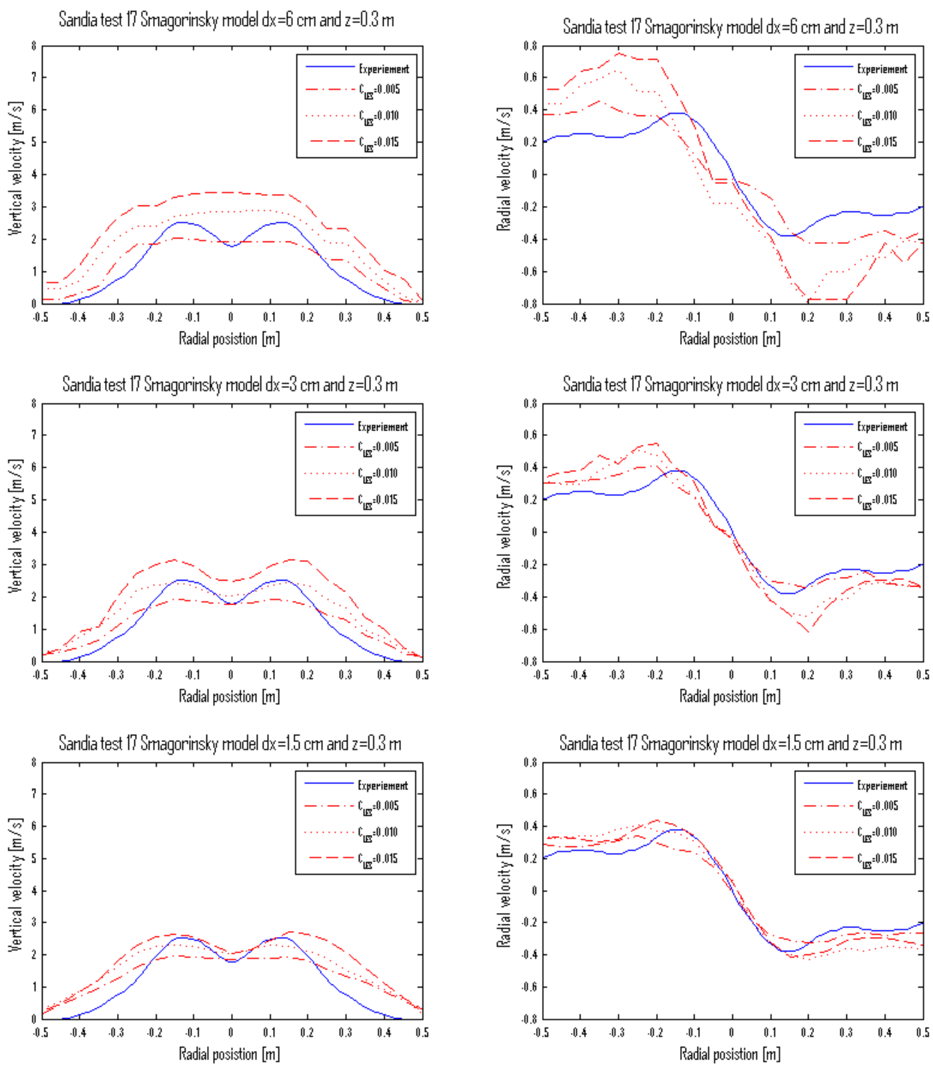


Figure 5.35: Sandia plume experiment test 17 with Smagosinky turbulence model at z = 0.3 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

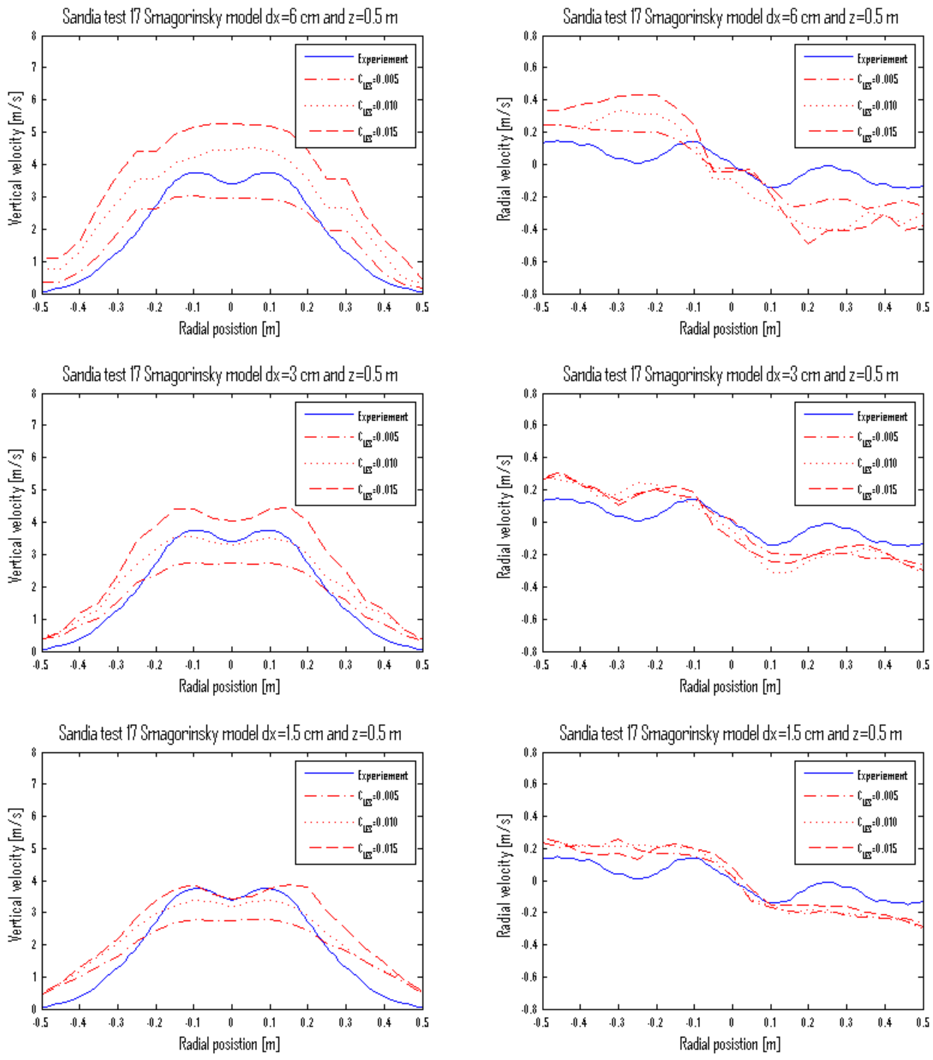


Figure 5.36: Sandia plume experiment test 17 with Smagosinky turbulence model at z = 0.5 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

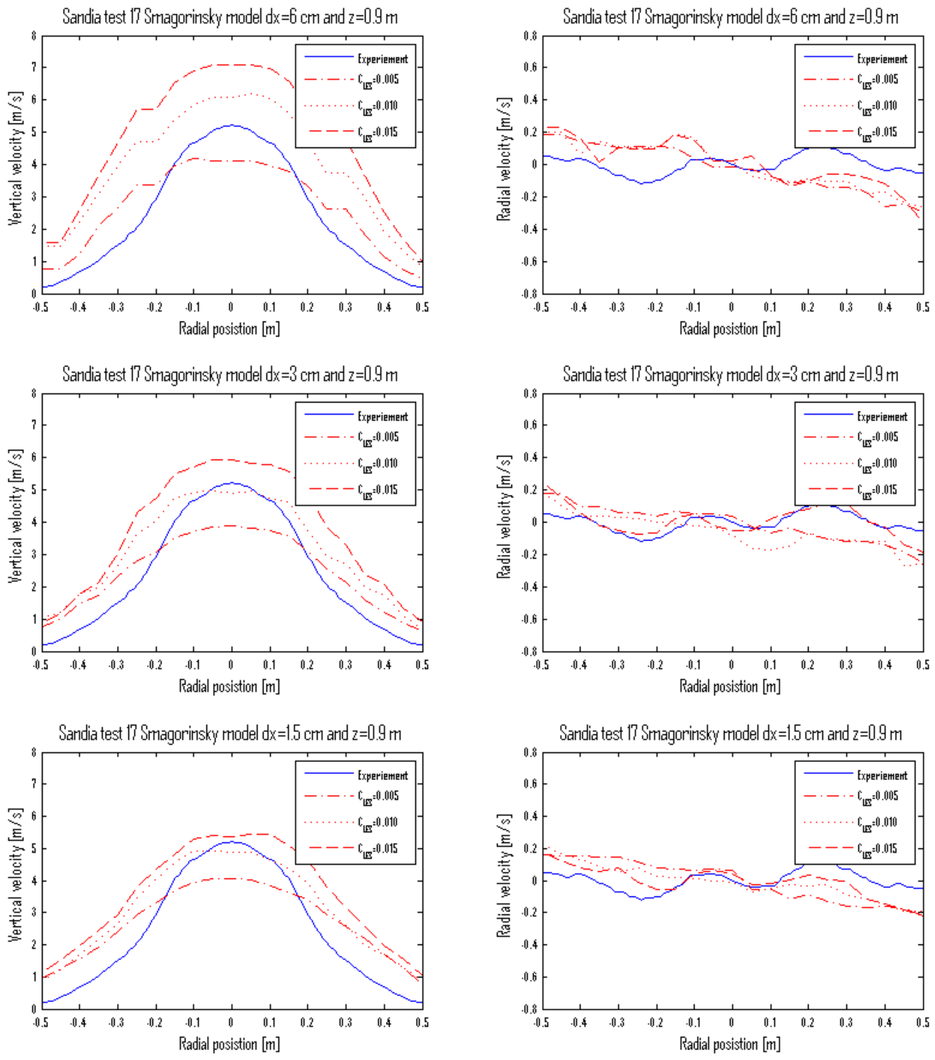


Figure 5.37: Sandia plume experiment test 17 with Smagosinky turbulence model at z = 0.9 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

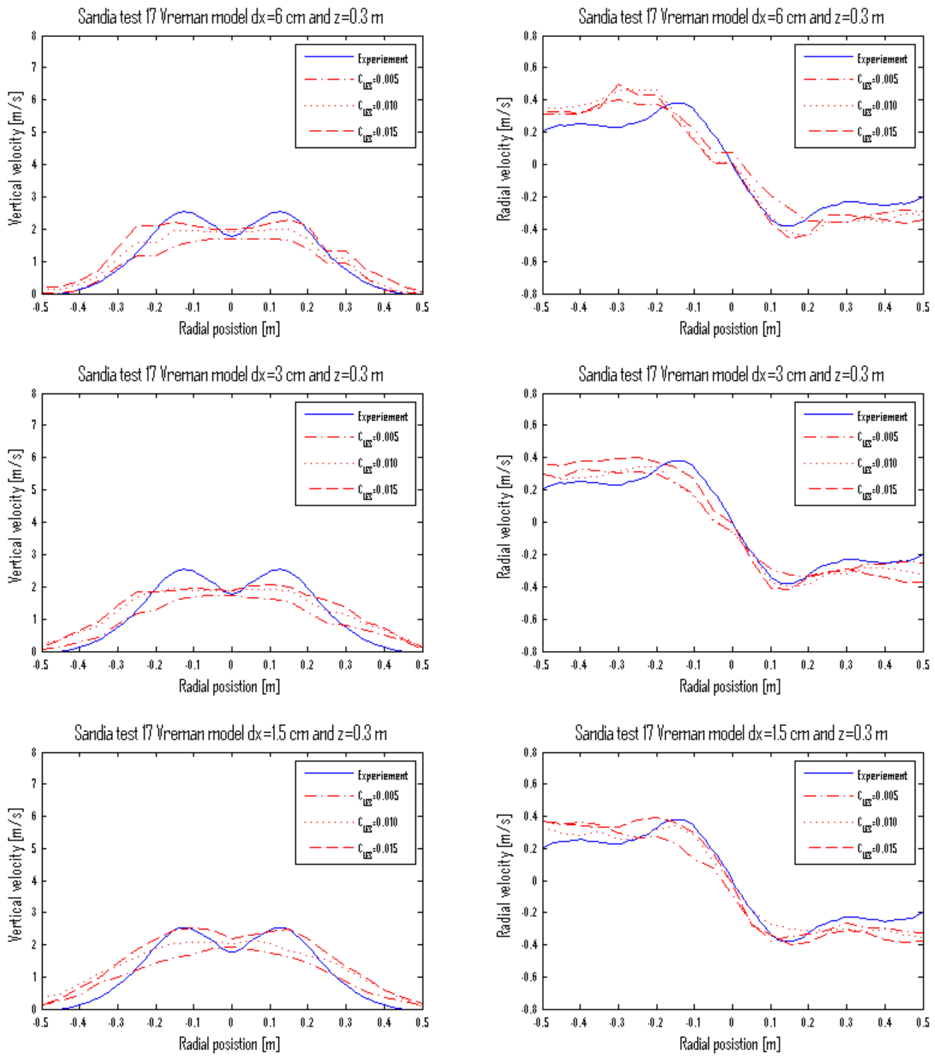


Figure 5.38: Sandia plume experiment test 17 with Vreman turbulence model at z = 0.3 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

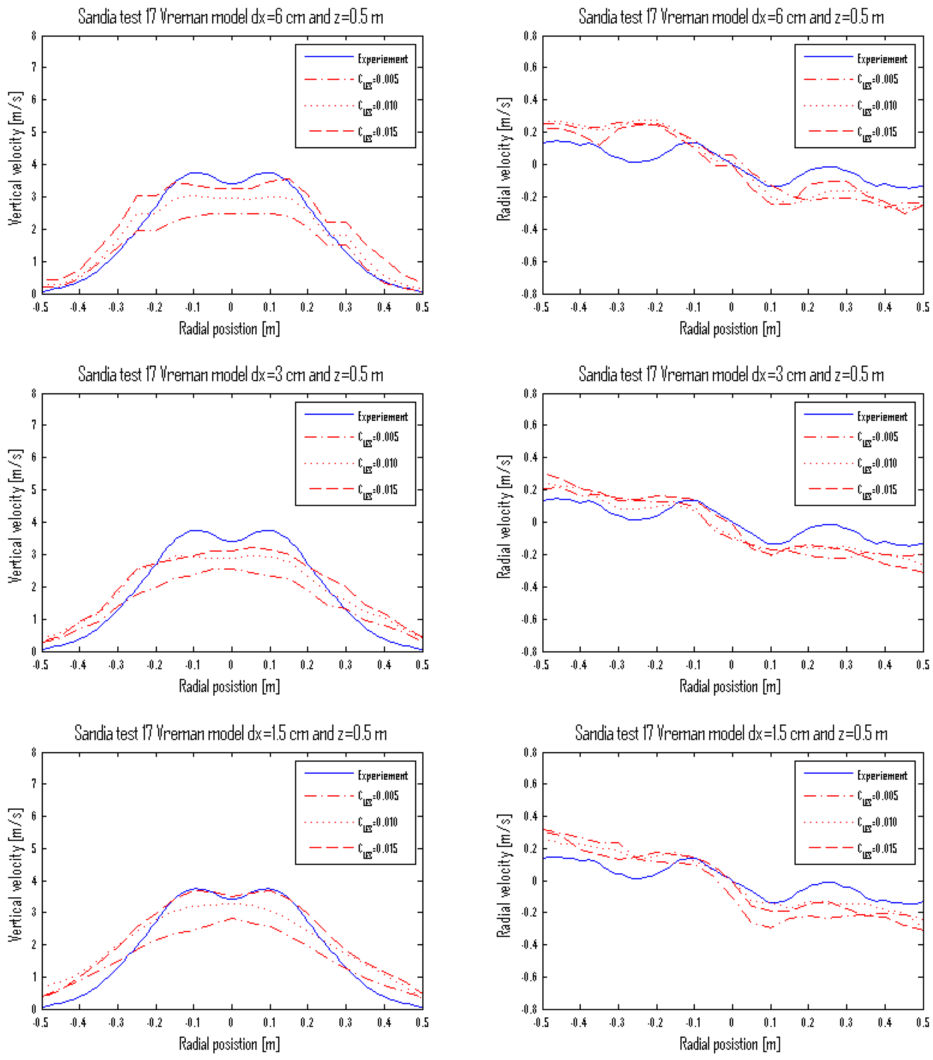


Figure 5.39: Sandia plume experiment test 17 with Vreman turbulence model at z = 0.5 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

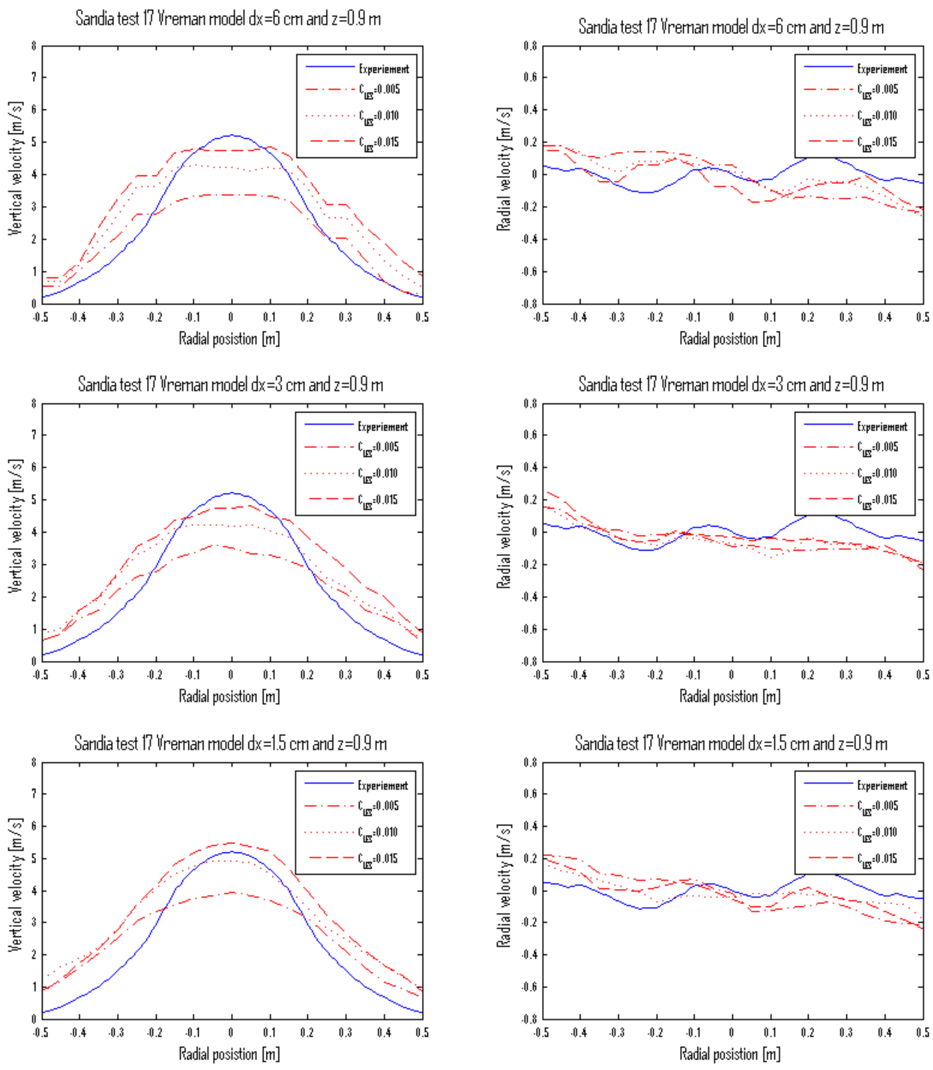


Figure 5.40: Sandia plume experiment test 17 with Vreman turbulence model at z = 0.9 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

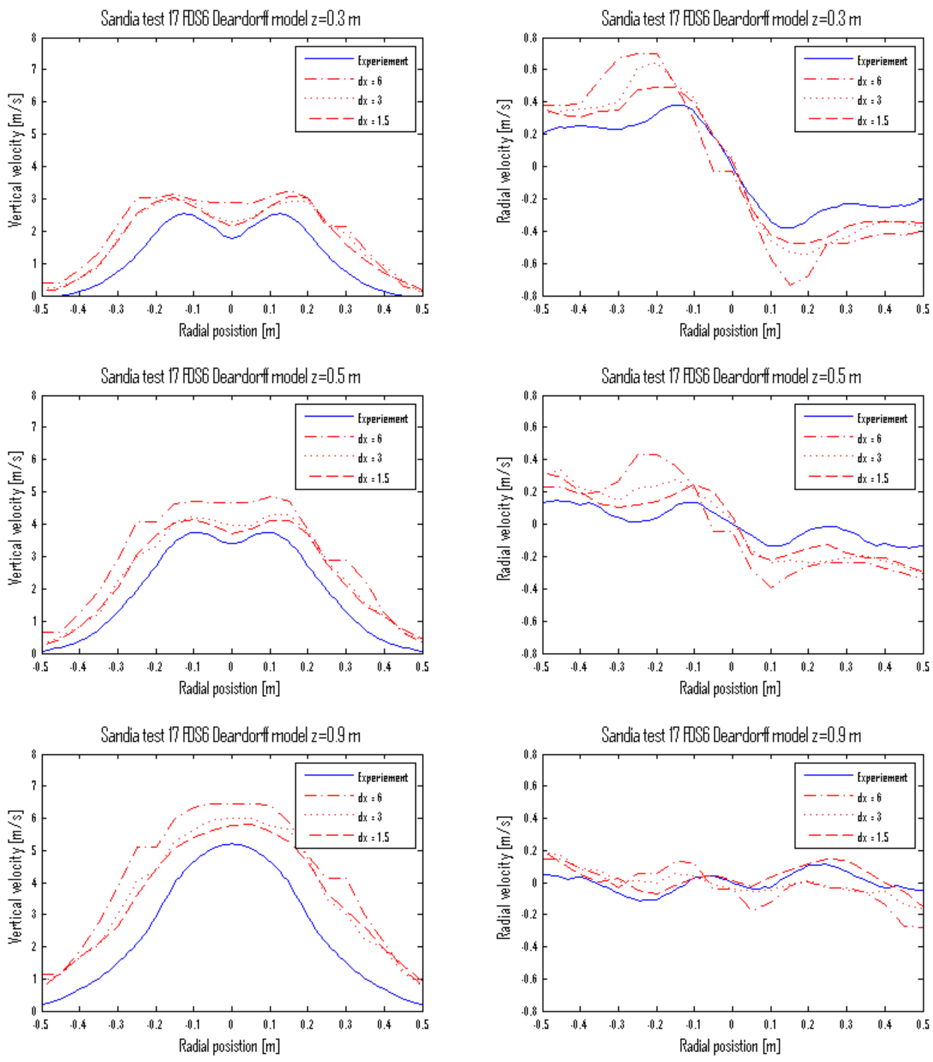


Figure 5.41: Sandia plume experiment test 17 with FDS6. Vertical velocity to the left and radial velocity to the right. On the top at z = 0.3 m, z = 0.5 m in the middle and z = 0.9 m at the bottom.

5.3.3 Methane Fire, Test 24

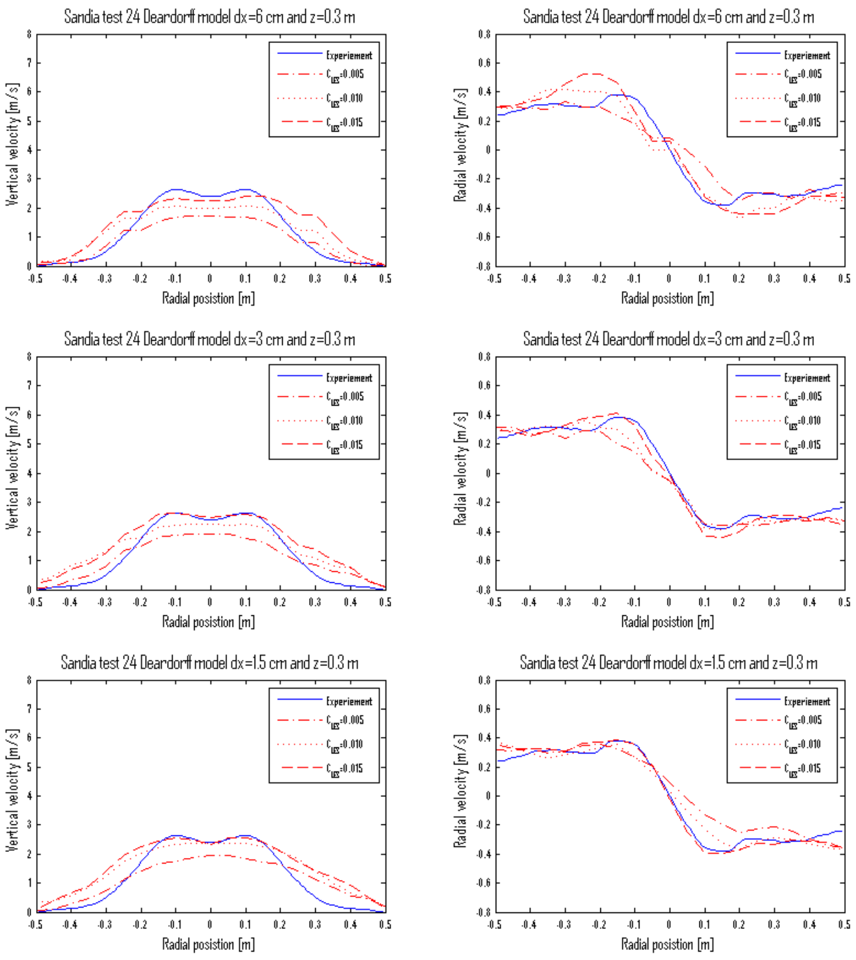


Figure 5.42: Sandia plume experiment test 24 with Deardorff turbulence model at z = 0.3 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

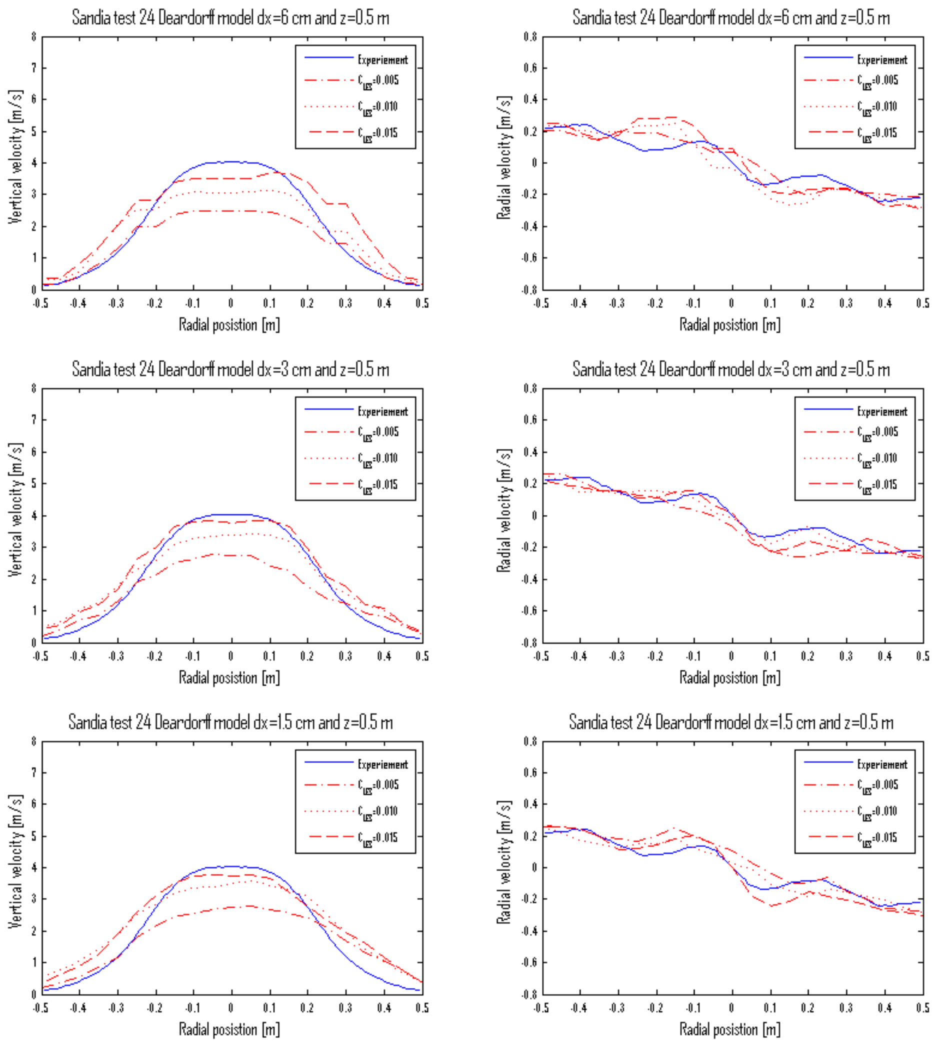


Figure 5.43: Sandia plume experiment test 24 with Deardorff turbulence model at z = 0.5 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

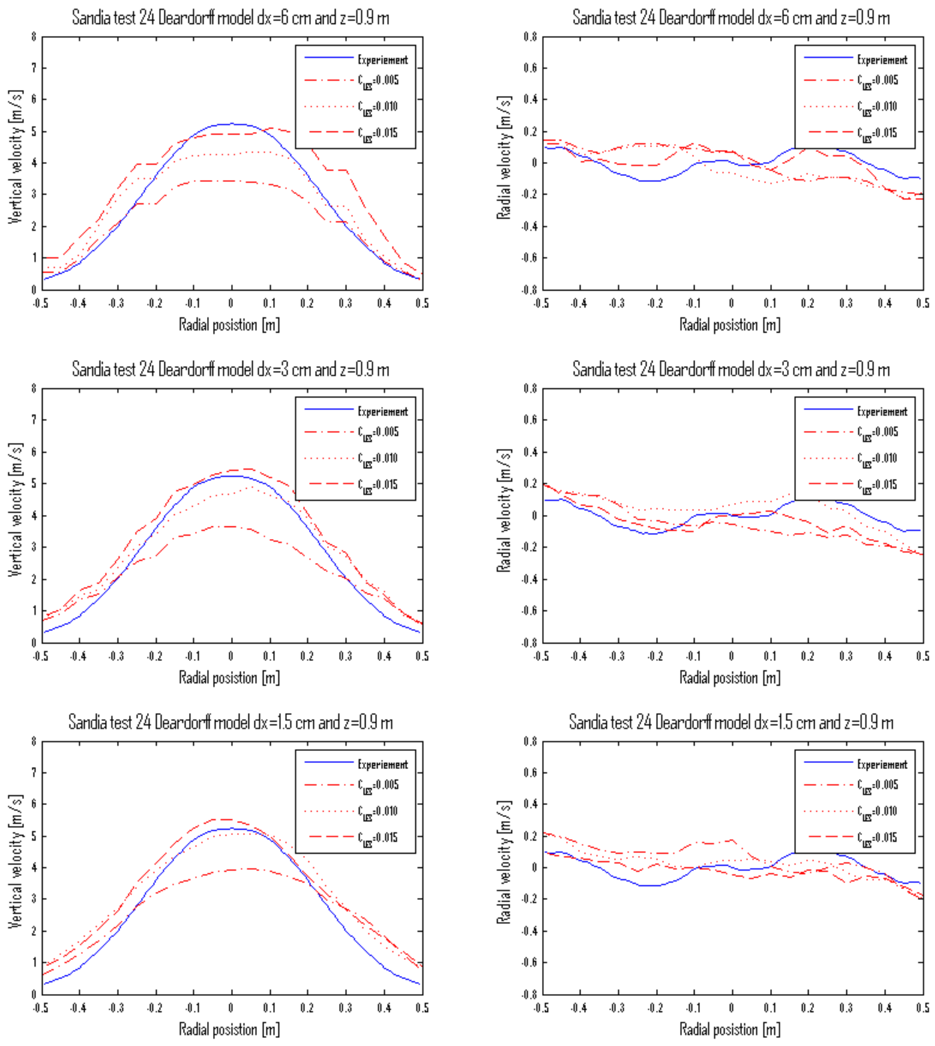


Figure 5.44: Sandia plume experiment test 24 with Deardorff turbulence model at z = 0.9 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

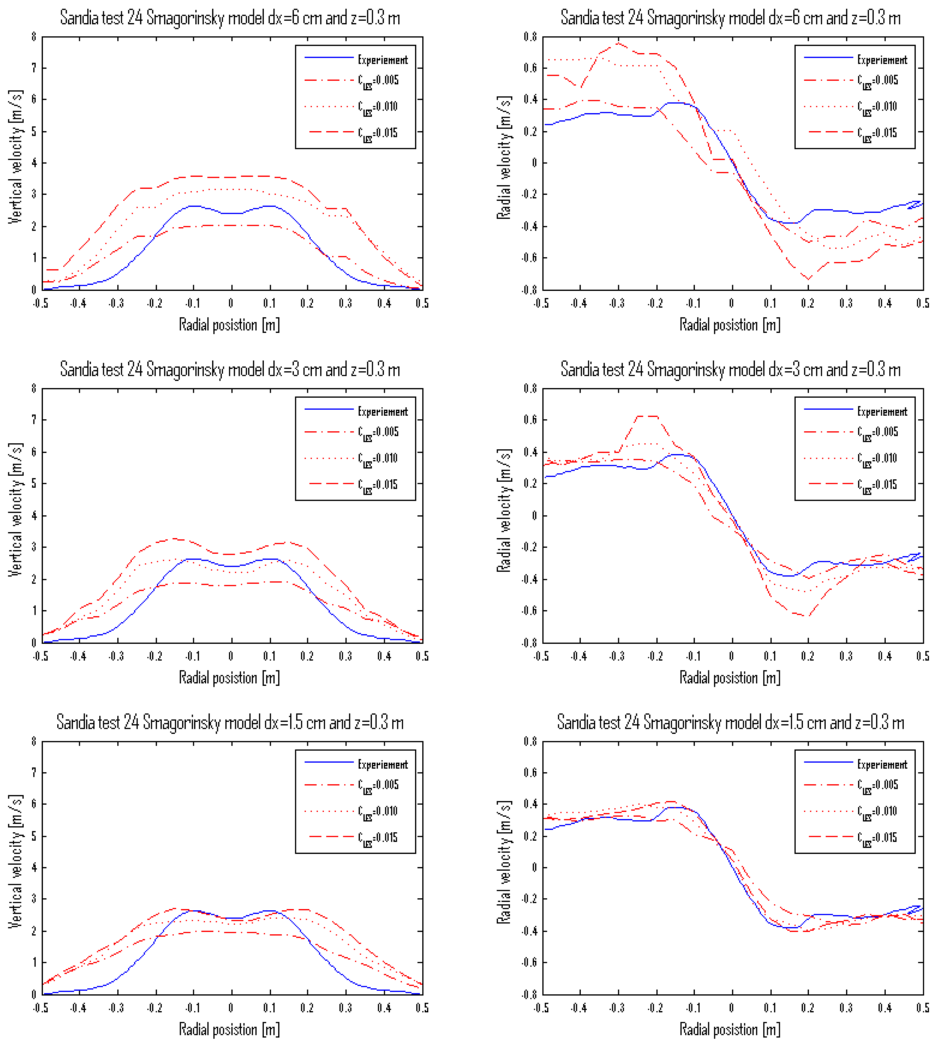


Figure 5.45: Sandia plume experiment test 24 with Smagosinky turbulence model at z = 0.3 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

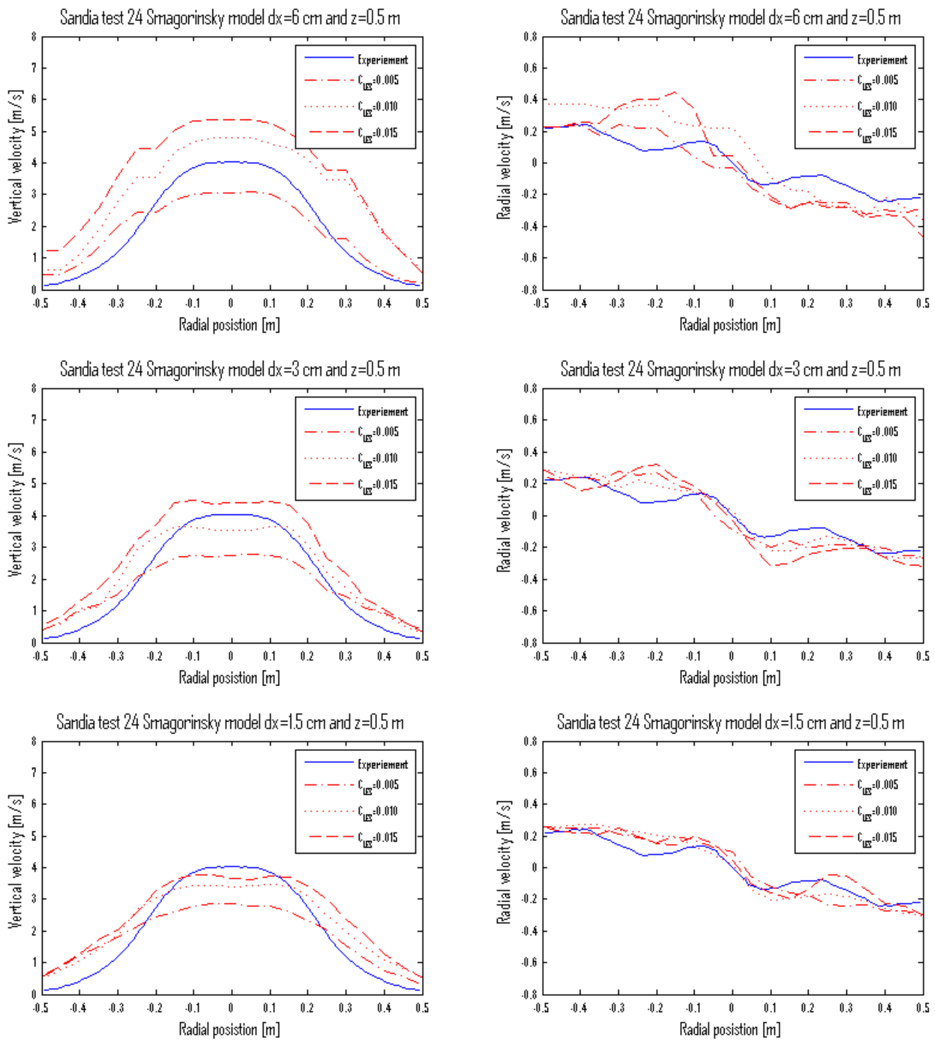


Figure 5.46: Sandia plume experiment test 24 with Smagosinky turbulence model at z = 0.5 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

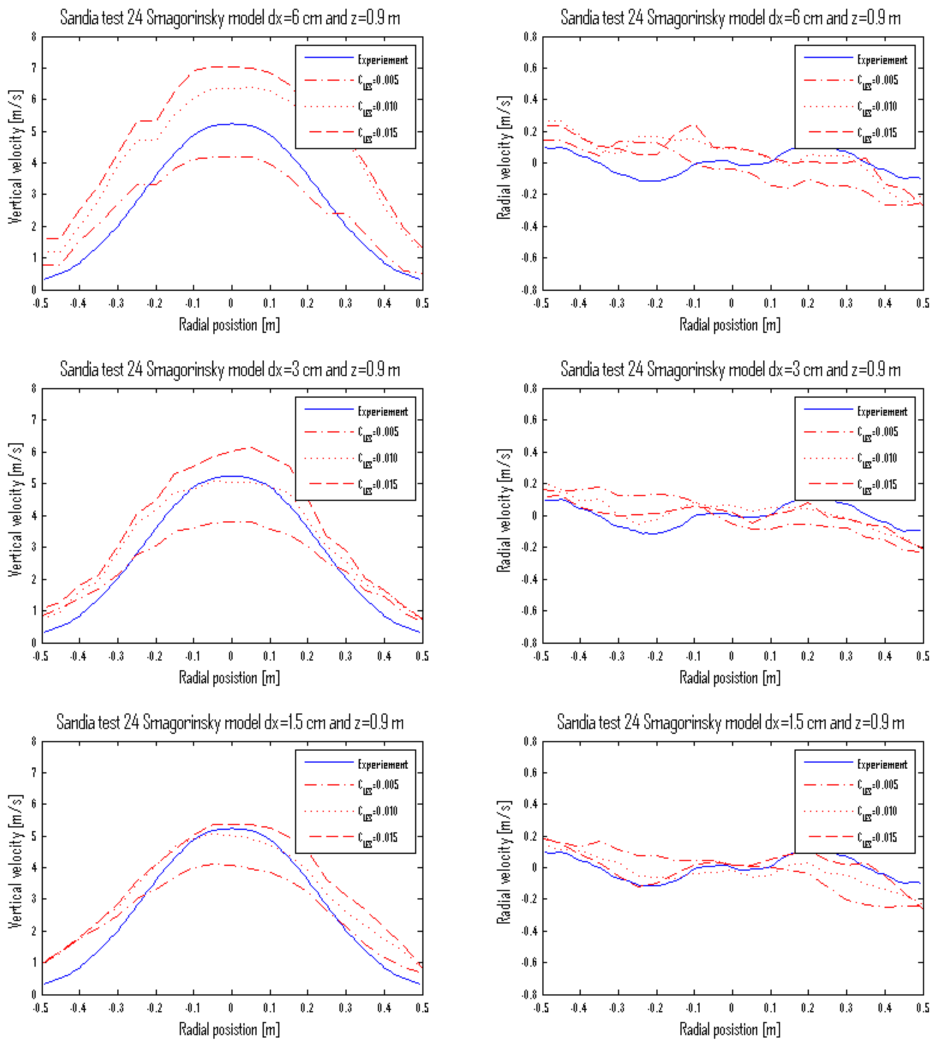


Figure 5.47: Sandia plume experiment test 24 with Smagosinky turbulence model at z = 0.9 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

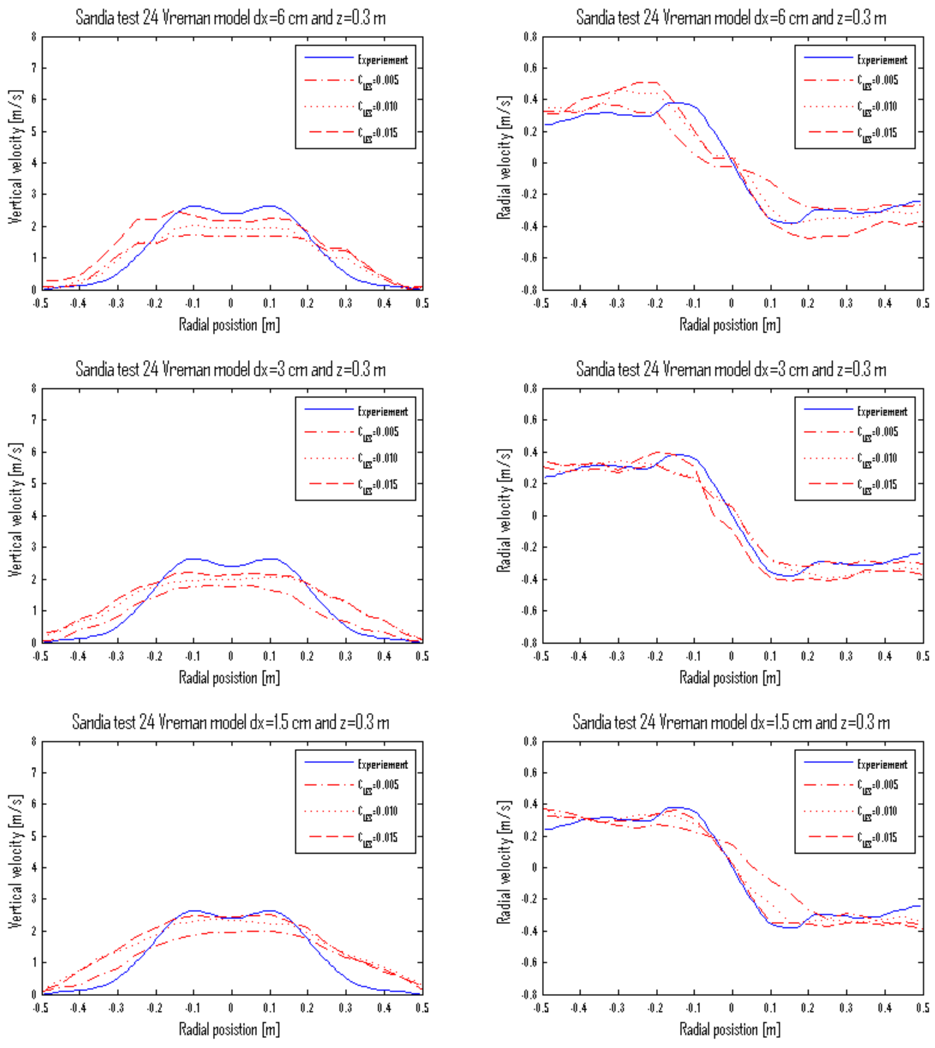


Figure 5.48: Sandia plume experiment test 24 with Vreman turbulence model at z = 0.3 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

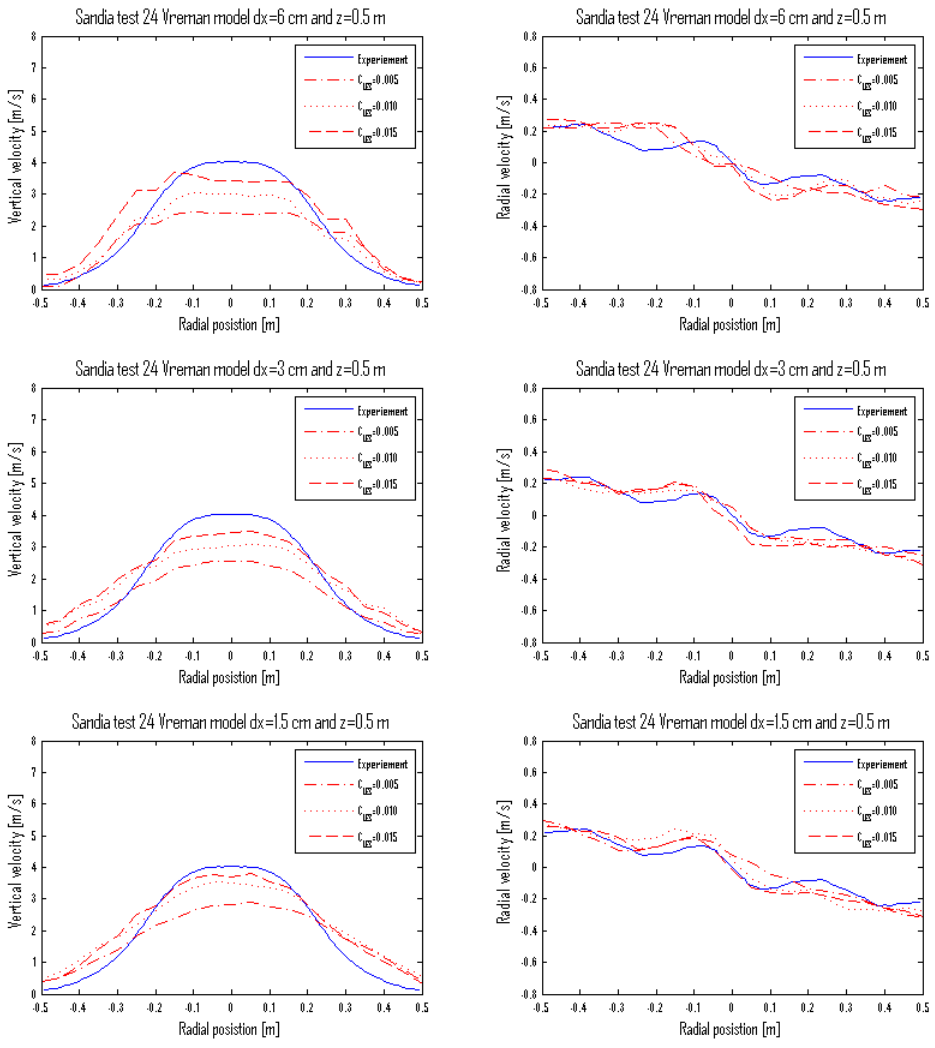


Figure 5.49: Sandia plume experiment test 24 with Vreman turbulence model at z = 0.5 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

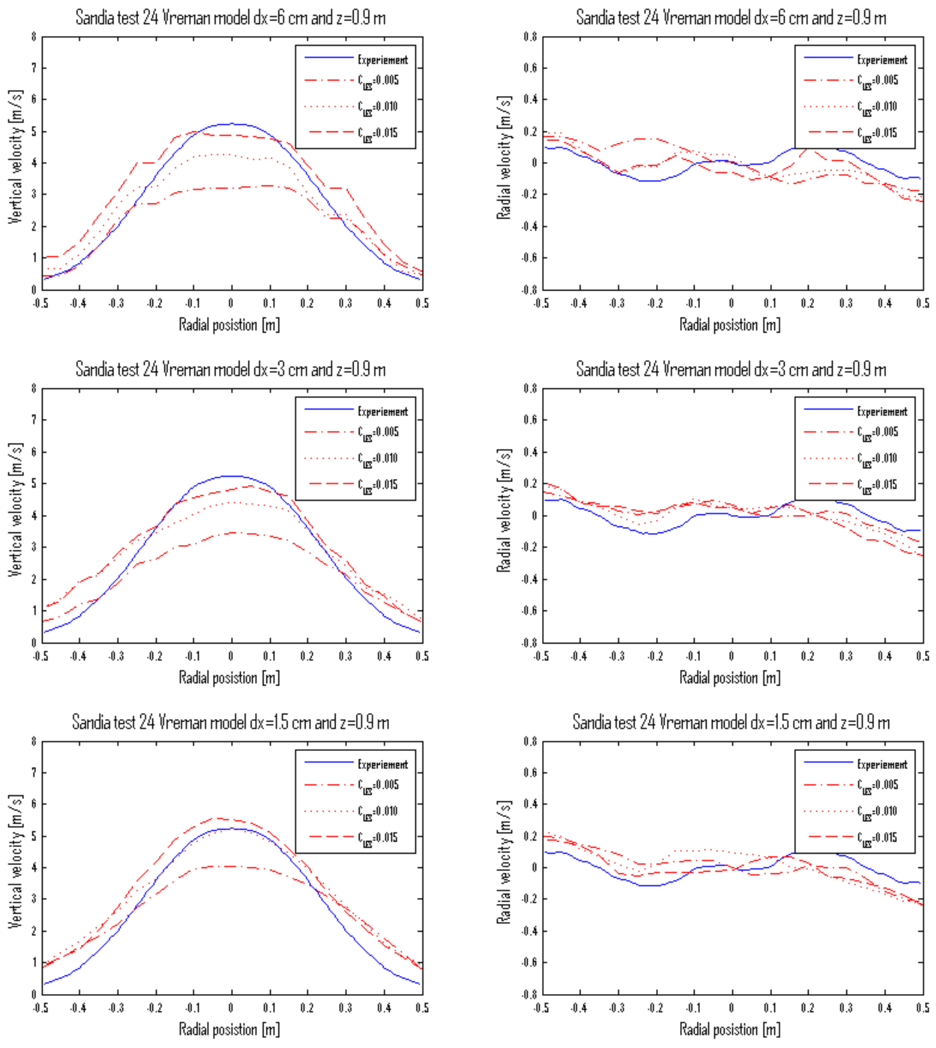


Figure 5.50: Sandia plume experiment test 24 with Vreman turbulence model at z = 0.9 m. Vertical velocity to the left and radial velocity to the right. On the top the coarse grid dx = 6 cm, dx = 3 cm in the middle and dx = 1.5 cm at the bottom.

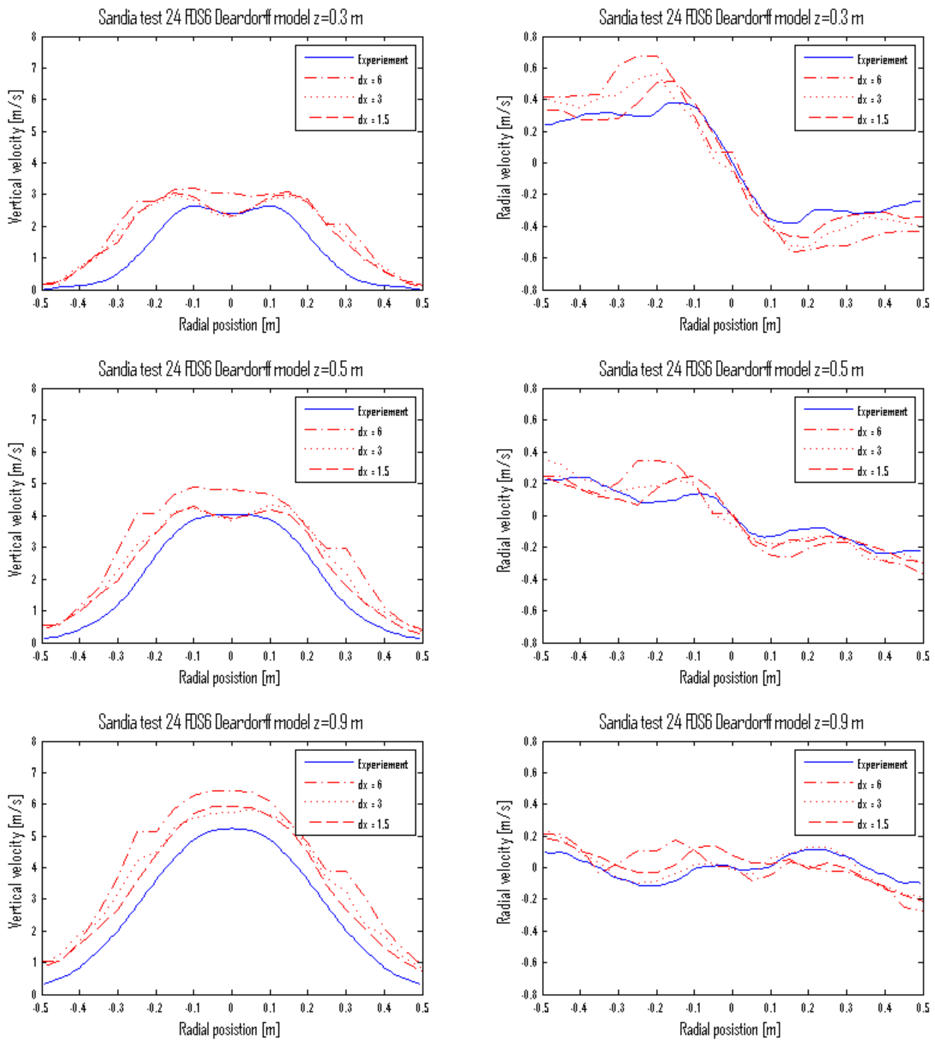


Figure 5.51: Sandia plume experiment test 24 with FDS6. Vertical velocity to the left and radial velocity to the right. On the top at z = 0.3 m, z = 0.5 m in the middle and z = 0.9 m at the bottom.

5.3.4 CPU clock time

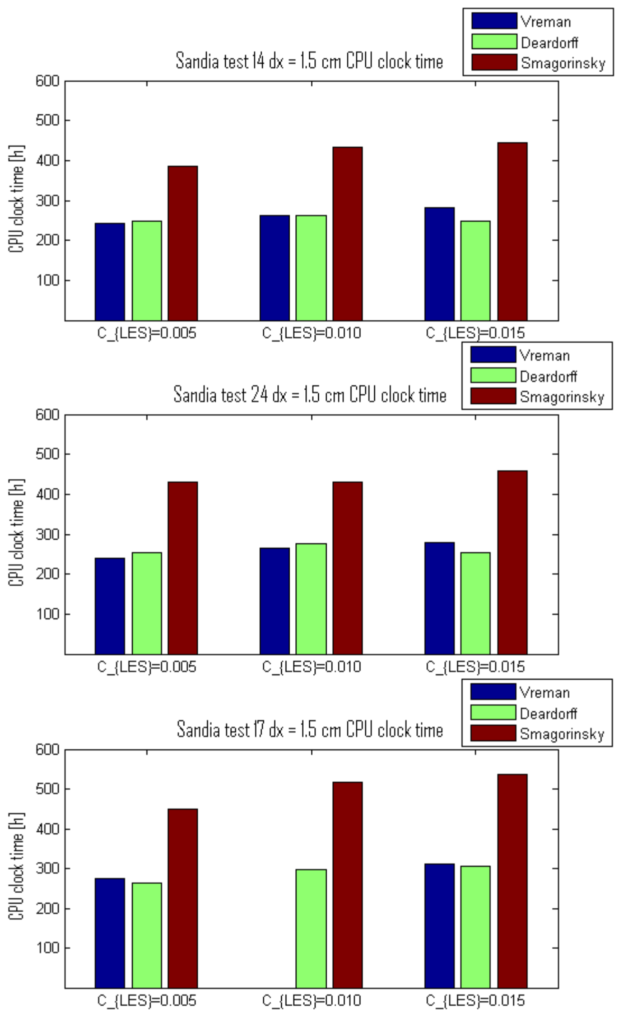


Figure 5.52: From top to bottom; CPU clock time for Sandia test 14, test 24 and test 17 with $dx = 1.5$ cm. Divided in groups of C_{LES}.

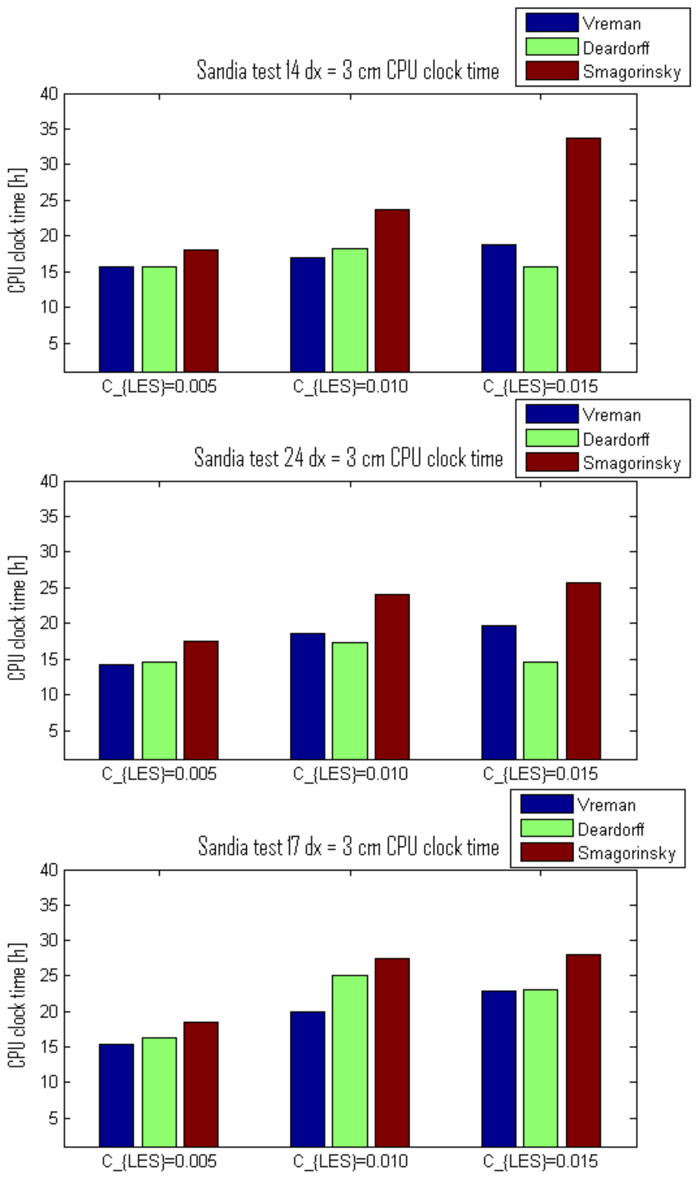


Figure 5.53: From top to bottom; CPU clock time for Sandia test 14, test 24 and test 17 with $dx = 3$ cm. Divided in groups of C_{LES}.

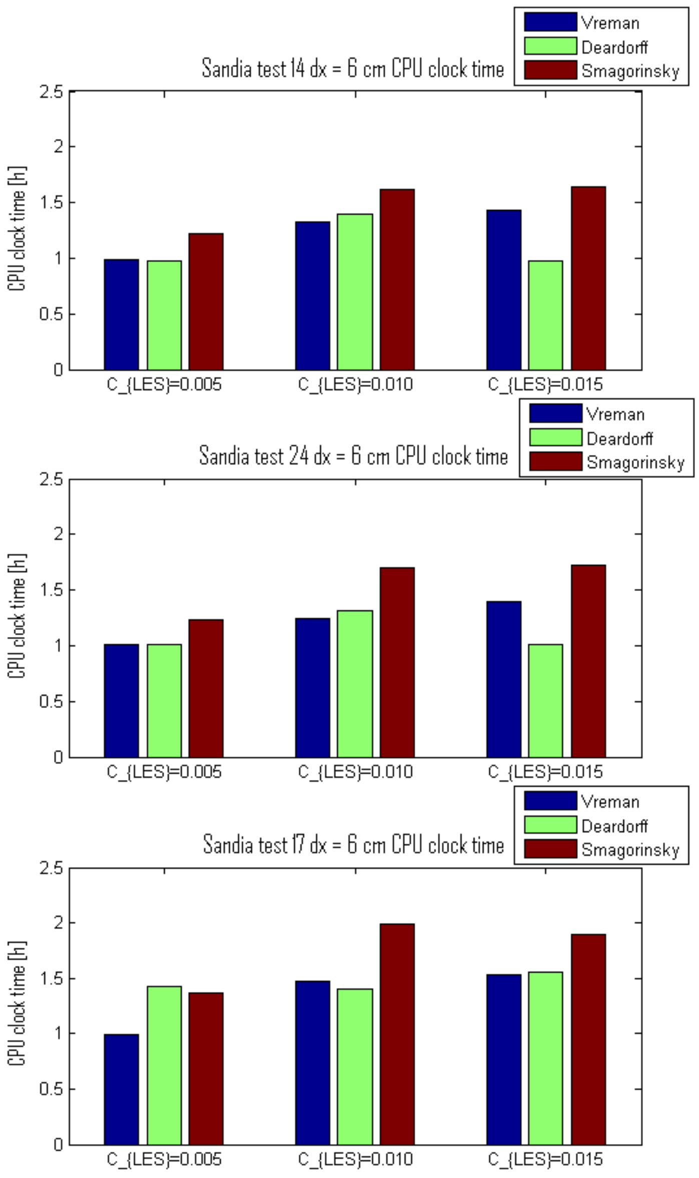


Figure 5.54: From top to bottom; CPU clock time for Sandia test 14, test 24 and test 17 with $dx = 6$ cm. Divided in groups of C_{LES}.

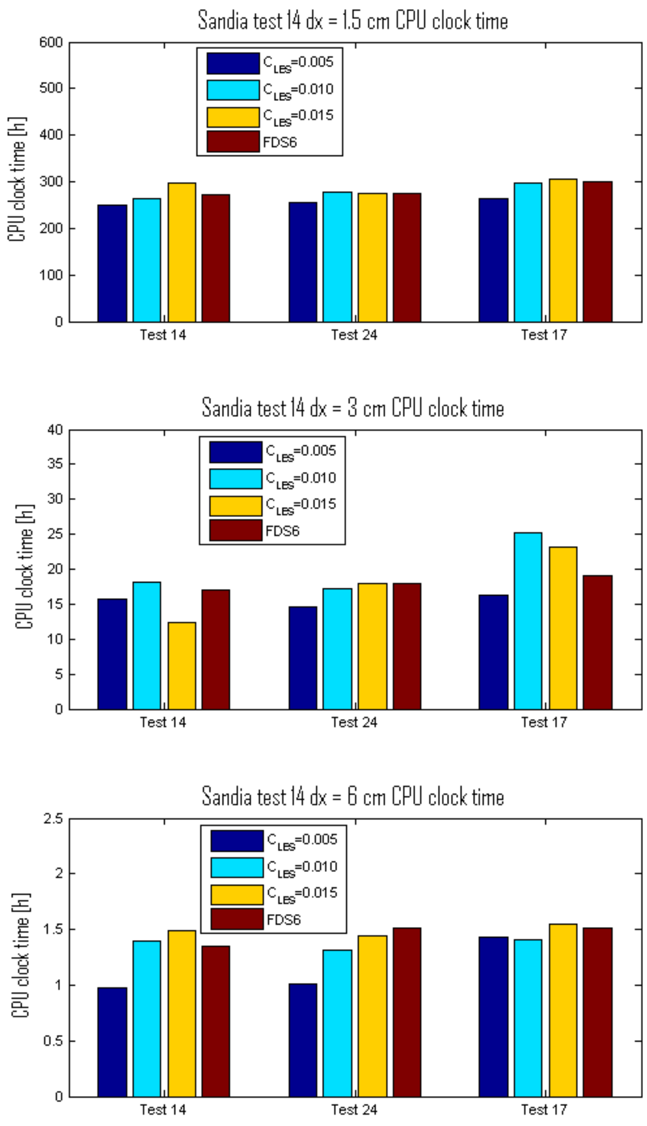


Figure 5.55: Comparison of CPU clock time between FDS6 and FDS-EDC for Sandia simulations. At the top $dx = 1.5$ cm, in the middle $dx = 3$ cm and $dx = 6$ cm at the bottom. Divided in groups of test 14, test 24 and test 17.

5.4 Discussion

5.4.1 McCaffery's Plume Correlation, Velocity and Temperature Profiles

In the McCaffery simulation the connection between centerline temperature and velocity are investigated when C_{LES} and grid resolutions are varied. With the fine grid the velocity profile is acceptably predicted with all turbulence models for all C_{LES}. The main difference is the peak value which is under estimated for $C_{LES} = 0.005$. However, the temperature is strongly under predicted just above the burner ($z < 1$) for $C_{LES} = 0.005$. The lower region right above the burner, approximately below 1 m where the velocity and temperature peak i located, is the most challenging area to model correctly. Particularly the peak temperature and its location. It is meaningless to quantify the error for the the temperature profile because of the steep slope. The error is large in this area, but the curves have the same shape and are just displace. For all the turbulence models, the main difference in change of $C_{LES} = 0.01$ to $C_{LES} = 0.015$ is the maximum temperature.

Simulations with Deardorff turbulence model show that $C_{LES} = 0.005$ fits best for the velocity profile for 14 kW and 22 kW (5.1 and 5.2). As the HRR is increased $C_{LES} = 0.01$ and $C_{LES} = 0.015$ gives better results for the velocity even though the velocity peak is over estimated. With the finest grid, the temperature peaks are located too far away from the burner and becomes larger as the HRR is increased. This displacement is largest for $C_{LES} = 0.005$. For the other C_{LES} the error is about 50-70%. But for $C_{LES} = 0.015$ the peak value are strongly over predicted with up to 600°C (57 kW in Figure 5.5). Because of the over prediction, $C_{LES} = 0.01$ is the most correct constant for the Deardorff model in total.

The Smagorinsky turbulence model predicts the temperature profiles in McCaffery simulations better than Deardorff and Vreman. The temperature profile is in excellent agreement for $C_{LES} = 0.01$ with the finest grid. The under predictions of the velocity profiles are larger with the Smagorinsky model, particularly for the two highest HRR of 45 kW and 57 kW. However, the error in maximum temperatures for both $C_{LES} = 0.01$ and $C_{LES} = 0.015$ are not as over predicted as for the other models. The results for velocity is almost identical for the coarse grid when C_{LES} is changed.

The errors in maximum temperature for the Vreman turbulence model with $C_{LES} = 0.015$ are the smallest. With $C_{LES} = 0.01$ the errors are about the same as for the other models.

In Figures 5.16 and 5.17 the displacement of temperature peak value and the location for FDS6 may be seen. The maximum temperatures are not over estimated as much as for FDS-EDC with $C_{LES} = 0.015$ (Deardorff) for the three scenarios with largest HRR. The under estimation of velocity profiles is slightly larger for FDS6 than for FDS-EDC in contrast to the peak value, which more correctly predicted.

In Figure 5.56, Temperature and velocity contours for McCaffery 57 kW with Deardorff turbulence model is presented. This figure can be seen in connection with Figure 5.58 where the contour plot of HRRPUV is seen. Even though the fire size not equal the principle is the same. For the lowest $C_{LES} = 0.005$ the HRRPUV is under estimated

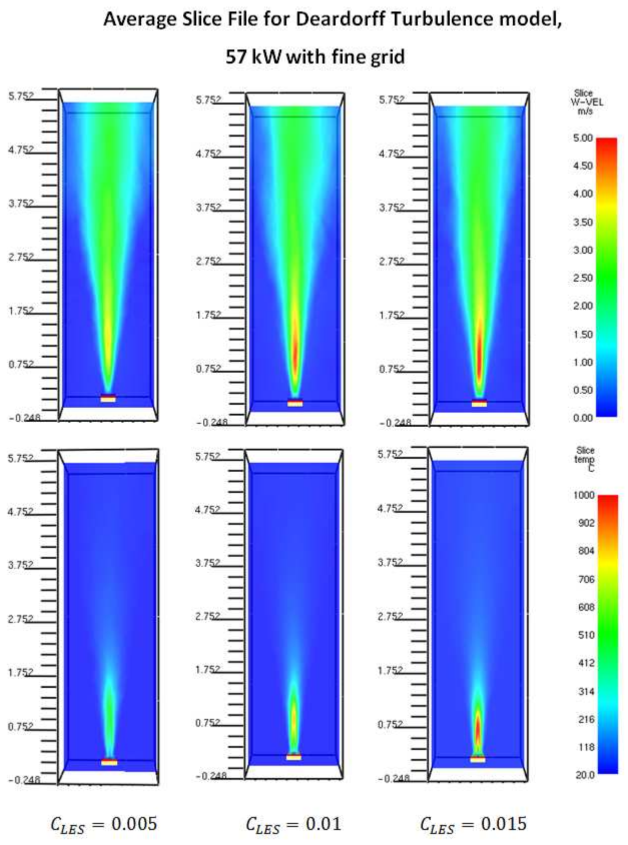


Figure 5.56: An example of temperature (bottom) and velocity (top) contour plot for McCaffery 57 kW with Deardorff turbulence model and the finests grid.

and smeared out over a longer travelled distance than the flame height. This leads to an under estimation of velocity and temperature. For the flame height $C_{LES} = 0.015$ is in best agreement with Heskestad correlation while the temperature and velocity profiles with $C_{LES} = 0.01$ is in best agreement with McCaffery correlation. I.e the same value of the constant do not necessarily has to fit for flame height, velocity and temperature profile, even for the same fire size.

5.4.2 Heskestad Flame Height Correlation

Figures 5.18-5.21 show that FDS-EDC for all turbulence models are strongly grid depended and are over predicting the flame height for coarse and medium grid resolution ($D^*/dx = 5$ and $D^*/dx = 10$). A constant can be established for all turbulence models to be in good agreement with the Heskestad correlation for large Q^* with the fine grid ($D^*/dx = 20$). Deardorff and Vreman turbulence model do not capture the slope for low Q^* as well as the Smagorinsky, especially not for the coarse grid. With the fine grid and

$C_{LES} = 0.015$, Vreman gives the worst results for $Q^* < 1$ with error ranging from -27% to 50% in contrast to Smagorinsky which captures the slope excellent. Except from the Smagorinsky model, the flame height is little influenced by the model constant for the coarse grid. However, the results are most sensitive to changes in C_{LES} for low Q^*.

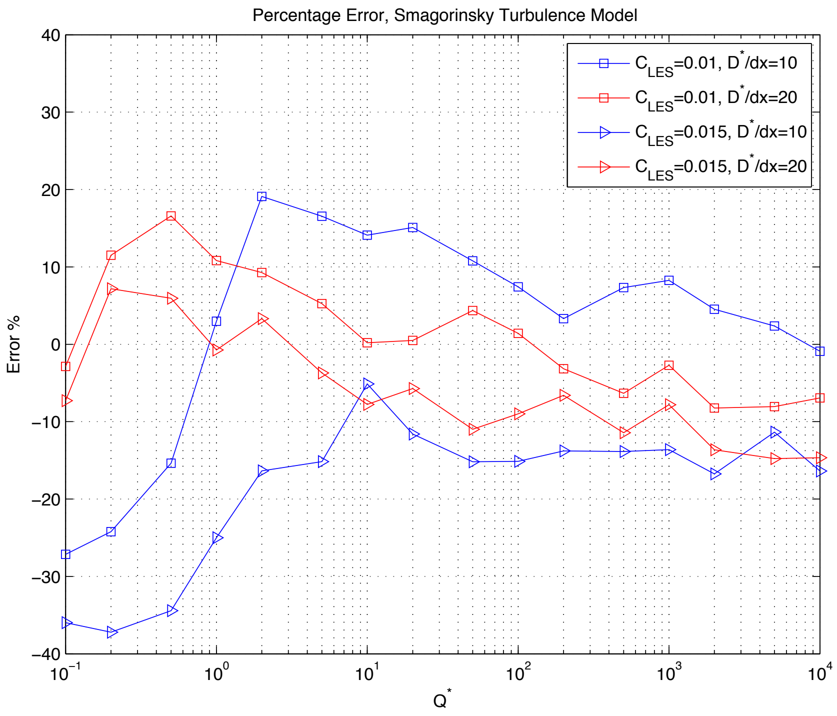


Figure 5.57: Comparison of percentages error in Heskestad flame height correlation with Smagorinsky turbulence model.

$C_{LES} = 0.015$ gives results in best agreement with the Heskestad flame height correlation for the Deardorff turbulence model (in Figure 5.18). With $D^*/dx = 20$ the maximum error is 27% for $Q^* < 1$ (expect when $Q^* = 0.5$) else it is 12%, and for $D^*/dx = 10$ the maximum error is 25% in the whole range of Q^*.

It is more difficult to establish a constant for the Smagorinsky turbulence model with the fine grid in Figure 5.19. The Smagorinsky model is the model that is most sensitive to changes in the model constant with $D^*/dx = 5$ and $D^*/dx = 10$. By plotting the percentage error as in Figure 5.57, it can be seen that both $C_{LES} = 0.015$ and $C_{LES} = 0.01$ gives excellent results for $D^*/dx = 20$, even for low Q^*. With $C_{LES} = 0.015$ the absolute error is less than 14.7%. For medium grid resolution $D^*/dx = 10$, $C_{LES} = 0.01$ predicts the overall flame height slightly better than $C_{LES} = 0.015$, particularly for $2 > Q^*$. Both constants under estimate the flame height for the lowest Q^* with medium grid resolution.

With $C_{LES} = 0.015$ for the Vreman turbulence model (Figure 5.20), the flame height

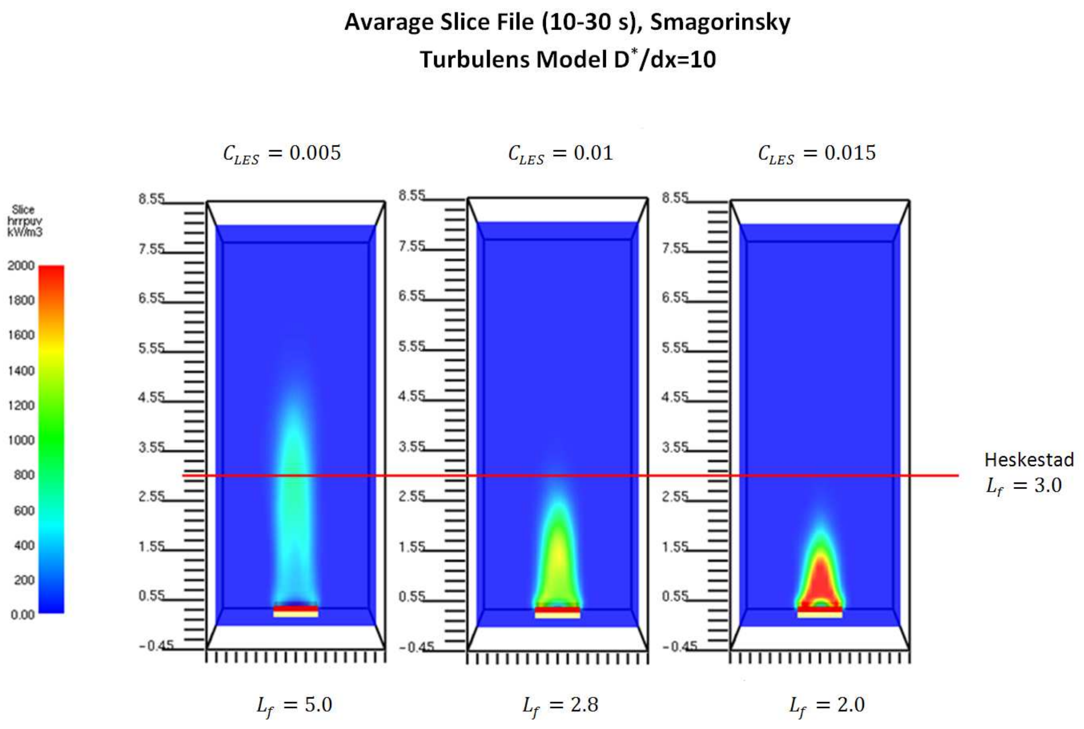


Figure 5.58: Slice file for heat release per unit volume (HRRPUV) versus flame height averaged from 10-30 seconds for Smagorinsky turbulence model, $D^*/dx = 10$.

deviates by -27% to +50% in the range $0.1 \leq Q^* \leq 2$ and up to 10% for $2 < Q^*$ with $D^*/dx = 20$. For the coarse grid, the flame height is over predicted by about 40%.

In Figure 5.21 FDS6 and FDS-EDC are compared with Heskstad flame height correlation for the Deardorff turbulence model. Surprisingly, FDS6 has also difficulties to capture the slope correctly in the interval $0.1 \geq Q^* \geq 1$. FDS6 is over predicting the flame height as FDS-EDC in this interval. In contrast to FDS-EDC, FDS6 is not as grid depended and gives satisfactory results for all grid resolutions. The explanation to this is that the mixing times are functions of the grid cell size, seen in eq. (3.4). The LES-EDC model could simply be made grid independent by establishing a dynamic model constant, C_{LES}, as a function of the grid cell size. For FDS6 the absolute error is less than 13.7% with $D^*/dx = 20$ if $2 < Q^*$, else it is up to 46%.

Actually, the absolute error i less for the coarser grids for low Q^*. With $D^*/dx = 10$ the 10.3% and 12.9% with $D^*/dx = 5$ except from $Q^* = 0.1$ (63% error).

Figure 5.58 is an example on how C_{LES} affects the flame height. Since the flame height is assumed to be where 99 % of fuel is consumed on average, a slice of heat release per unit volume (HRRPUV) is a good way to illustrate how C_{LES} affects the physics. When C_{LES} is increased, the local HRR also increase (right side of Figure 5.58). In case of under prediction of HRRPUV (small C_{LES}) too little fuel is consumed in the lower flame region and allows more fuel to raise due to buoyancy forces. As a result, the fuel moves a longer distance before it is consumed, hence flame length is over estimated (left side of Figure 5.58).

5.4.3 Sandia Plume

It is obvious from the results that a time average of 10 seconds is too short. Since the boundery conditions are symmetric it is reasonable to expect symmetric behavior around the centerline of the flame. The consequence of too short time average may be seen in test 14 with the Smagorinsky turbulence model for the coarse grid in the lowest positions $z = 0.3$ m and $z = 0.5$ m (Figures 5.25 and 5.26) where the vertical peak velocity is displaced away from the center in radial position. This misplacement is generally not observed in such extent for simulations with finer grid. Radial velocites are also influenced by the too short time average period because the velocities in few cases are zero along the centerline.

The results show that the trend is the same for all the turbulence models. Not surprisingly, the velocity profiles are more difficult to model correctly as the position increase away from the fuel source. This can especially be seen for the lowest HRR in test 14 in $z = 0.9$ m, Figures 5.24, 5.27 and 5.30. It is difficult to establish a single C_{LES} that fits all the cases in all positions. The vertical velocity is overestimated in the edges of radial position in all heights even for the lowest C_{LES} of 0.005. However, an error of the vertical velocities in the edges or the radial velocity (in high z position) is not that crucial for the flame behavior. The reason is that in these places the velocities are low, and even though the percentage error is large the contribution to the over all upward mass rate in the flame is low. Actually, the velocity profiles should be considered in connection with densities to ensure correct upward mass transport. Experimental results could maybe reveal that the velocity is overestimated in place where the density is underestimated (i.e that underestimation of density compensate for overestimation of velocity) so that the mass transport is correctly modeled.

Deviation of the vertical velocity between the C_{LES} is largest in simulations with the coarse grid. The deviations are larger as the z-position increase. Velocities are over predicted with Smagorsinky turbulence model and FDS6 cases with the coarse grid which is in agreement with theory in Section 3.1 (over prediction of HRR if the grid is too coarse).

The radial velocity seems to be more difficult to model correctly than the vertical velocity. In Figures 5.25, 5.35 and 5.45 the radial velocity for the Smagorinsky turbulence model in $z = 0.3$ m appears almost to be random with the coarse grid. All over, Smagorinsky is the most grid dependent turbulence model. The error in the edges for the vertical velocities are slightly larger with Smagorsinksy compared with the other two turbulence models. In general, this error is less at the edges for all the models in $z = 0.5$ m and $z = 0.9$ m where the slope is steeper. In contrast to Vreman and Deardorff turbulence models, simulations with Smagorinsky and FDS6 are capturing the dip in the fuel rich area along the centerline of the flame in test 17. But for some reason the vertical velocity for all simulations, even for FDS6 (with the original combustion model),

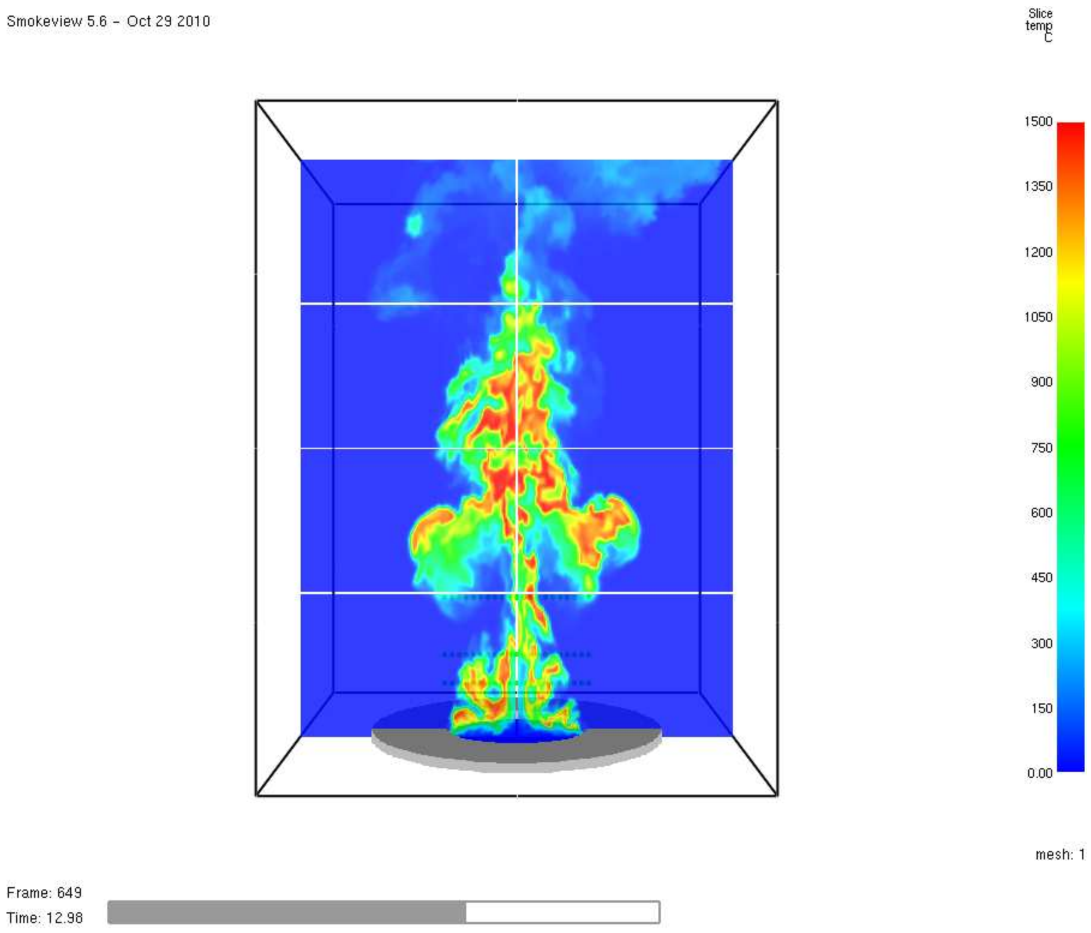


Figure 5.59: An example of temperature contours for FDS6 test 14 with $dx = 1.5$ cm at 13 second. Be aware that the screenshot represents instantaneous values and is not representative for the whole simulation time. The green dots is measuring points in heights of 0.3 m, 0.5 m and 0.9 m.

are remarkably over predicted 0.5 m and 0.9 m above the burner in test 14.

If FDS6 is used as acceptance criteria, all turbulence models for FDS-EDC with $C_{LES} = 0.015$ predicts the vertical velocity as well as FDS6 or even better with the finest grid. However, FDS6 is not as grid dependent as FDS-EDC. Other C_{LES} than 0.015 fits better in some cases with grids corresponds to $dx = 3$ cm and $dx = 6$ cm. But a constant should be establish for a fine grid ($dx = 1.5$ cm) and then in further work be modified to be grid independent. The deviation between $C_{LES} = 0.01$ and $C_{LES} = 0.015$ in the vertical velocity at $z = 0.3$ m is minimal in test 17 and 24 with all turbulence models. With $C_{LES} = 0.015$ the absolute maximum error of vertical velocities are approximately 15-20% while the radial velocity errors are some places up several hundred percent around the center .

Almost the same amount of code lines are added to the code as the amount of code lines which is commented out. Furthermore, one IF loop is commented out and one is added. If two similar cases is simulated with FDS6 and FDS-EDC contains the same amount of burning cells, then they should use the same CPU clock time in theory. Figure 5.55 confirms that this is almost correct. No systematic connection cannot be drawn whether CPU clock time for FDS6 simulations are larger than for FDS-EDC or the other way around. However, HRR release is related to the simulation time in most cases (see Table 5.2). Hence, test 17 takes longer time to simulate than test 24 and test 24 takes longer time to simulate than test 14. Figures 5.52 - 5.54 show that the Smagorinsky turbulence model is the most computational expensive. This is simply because the constant in the Smagorinsky model used in this thesis is dynamic and not static as for Deardorff and Vreman. The difference is much larger in percent if the grid i refined or the HRR increased. With the fine grid the difference is about 60%. Vreman and Deardorff turbulence models are roughly the same computational expensive.

Chapter 6

Flow Field Above Obstacle Inserted in Fire Plume

These experiments are inspired by experiments done by Lars Roar Skarsbøin a master's thesis delivered June 2011 at the University of Bergen [35]. Some improvements are done to make it closer to how it is modeled. In these experiments, square pipes are used instead of circular. The first reason is to generate more turbulence. Secondly, because a rectangular grid is used in simulations. Since only the combustion model is evaluated in this thesis and not the evaporation/pyrolysis and thermal radiation, liquid is replaced with gas to control the HRR. The burner size is decreased to be able to increase the flow velocity and also the degree of turbulence at lower HRR with less fuel.

The motivation for these experiments is to study the flow field above turbulence generating obstacles inserted in a fire. By systematically increase the HRR and height of the obstacles, a simple correlation between a turbulence property (TKE, turbulence intensity, velocity fluctuations, strain rate, etc.) and a dynamic constant C_{LES} may perhaps be established. It is also reasonable to believe that in such turbulent regime is where the largest difference between the sophisticated LES-EDC model and the already existing combustion model in FDS.

To finance the experiments, this study was linked up to the research program *Prediction and validation of pool fire developed in enclosures by means of CFD models for risk assessment of nuclear power plants.* During the planning, it turned out to be quite hard to find an institution that had the correct equipment and knowledge to operate Particle Image Velocimetry (PIV) to measure a velocity vector field. After some heavy delay, the experiments were performed at Lund University. Unfortunately, the outcome was no successful results. Hence, this study must be regarded as a pre-project for PIV measurements in the pool fire research programme and not a part of the validation of the combustion model.

6.1 Designing Experiments for CFD Validation

The philosophy of CFD model validation in matter of fire modeling may be separated in whether the fire is *specified* or *predicted* [32]. The choice is depended on the goal of validation. In this thesis the goal is to validate the combustion model, other models as evaporation, pyrolysis, soot, etc. is not of interest. Therefore, the fire should be *specified* rather than *predicted*. The road map for a CFD model in Figure 6.1 shows that the combustion is depended on other sub models in fire modeling. Sub models are only *models* consisting empirical correlations only valid in certain interval of physical values, and not necessarily physical formulas solving the actual physical problem. By specifying the fire the HRR is not influenced by quantities as radiation. Possible errors in sub models controlling the HRR are then eliminated. So in cases when single sub models are validated, it is preferable to involve a minimum of other sub models.

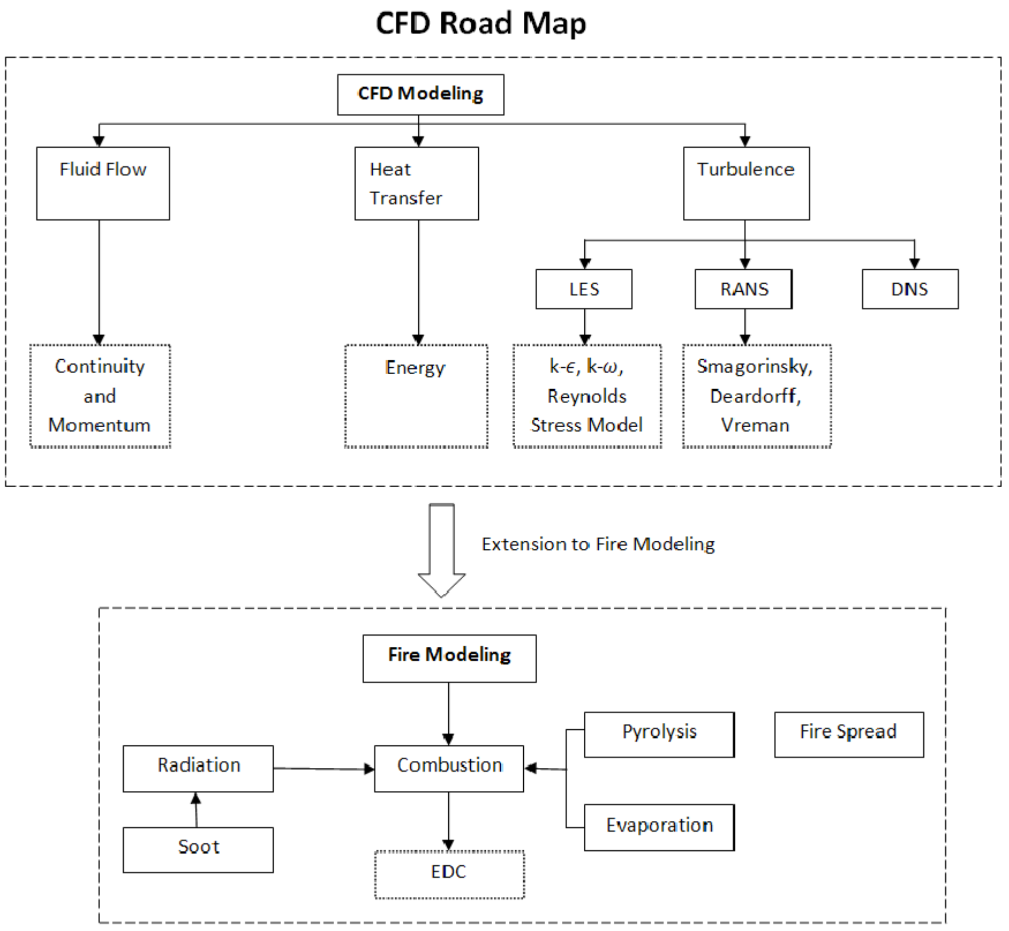


Figure 6.1: CFD road map; extension of CFD to fire modeling and overview of sub models.

In validations of CFD models, the experiments must be as close to the simulated scenario as possible. The HRR in the experiments must be controlled to be able to specify it in the input of simulations. A constant gas fire is therefore a good choice for validation of the combustion model. The HRR is then not influenced by the radiation which would be the case if a liquid fuel fire or a solid fuel fire is chosen. If a liquid fuel fire or a solid fuel fire is chosen, an error in the radiation modeling would influence the rate of pyrolysis or evaporation which further would lead to overestimation or underestimation of HRR.

Since uniform rectangular grid cells are applied in FDS, the geometrical shapes in the experimental set up should also be rectangular. So in these experiments, circular pipes used in Skarsbøs thesis were replaced with rectangular pipes. Additionally, geometrical dimensions smaller than the grid cell size should be avoided. However, most commercial CFD codes supports this by setting a porosity parameter in all faces of the grid cell.

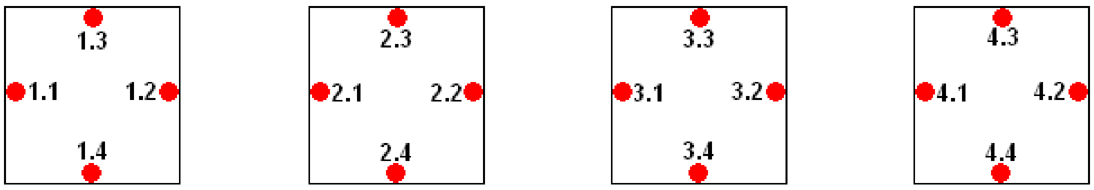


Figure 6.2: Numbering of thermal couples in the pipes.

Insertion of obstacles in flames leads to heat loss in the flame and affects the temperature, velocity, reaction rate, etc. Input in the simulations does not require inner temperature of the pipes, but it is an advantage to specify the temperature rather than predict it to eliminate a possible error in the heat transfer calculation. Therefore, 16 thermocouples were mounted in the pipes with improvised v-shaped wedges to find the steady state temperature. The placements of the thermocouples and numbering are seen in Figure 6.2.

6.2 Measurement Techniques

6.2.1 Particle Image Velocimetry (PIV)

Particle Image Velocimetry (PIV) is a sophisticated measuring technique for measurements of instantaneous velocity vectors in cross-section of a fluid flow. Wind tunnel velocity experiments, experimental verification of CFD models, measurements in pipe flows, spray and combustion research are examples of application areas of PIV [37]. Normally, low-mass particles are seeded in the flow, which is assumed to move free with the local flow velocity [36]. Two short time laser pulses illuminate a plane in the flow. Light is scattered by the seeded particles and are recorded by a synchronized camera. Through the post-processing, the two images are subdivided in *interrogation areas* and

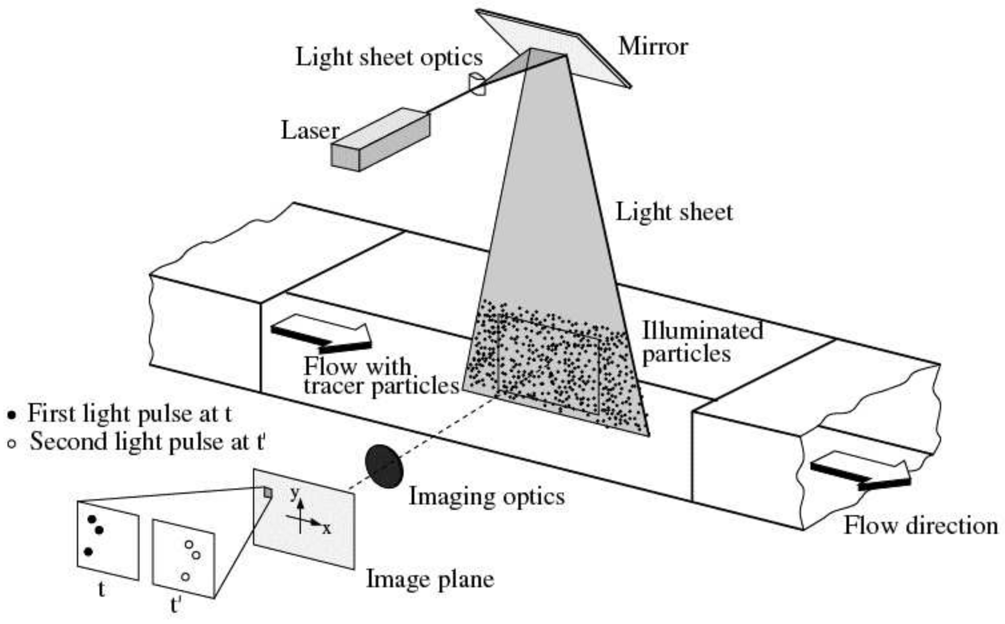


Figure 6.3: The principle of Particle Image Velocimetry (PIV) measuring technique [36].

the velocity vector is found from the movements of the particles. I order to obtain good signals, 10-25 particles should be recorded in each interrogation area [37].

6.2.2 Thermocouples

A thermocouple is a device for temperature measurements. Thermocouples can operate in a broad range of temperature and are inexpensive, durable and easy to apply. Two different conductors are coupled to a voltage logger, se Figure 6.4. The voltage produced by the conductors are proportional to the temperature. Temperature at the cold junction must be known and preferably kept constant to calculate the temperature at the hot junction. The temperature difference is given by

$$\Delta T = \sum_{n=0}^{N} a_n v^n \tag{6.1}$$

where v is the output voltage and a_n the a coefficient depending upon the metal. In some cases the temperature is found from a database. Databases are usually implemented in modern logging softwares on computers.

In the experiments in this thesis, thermocouples type K was used. For type K metal 1 is chromel (90 % nickel and 10 %chromium) and metal 2 amuel (95 % nickel, 2 % manganese, 2% aluminum and 1 % silicon). Thermocouples type K measures in the range -200 °C to 1350 °C.

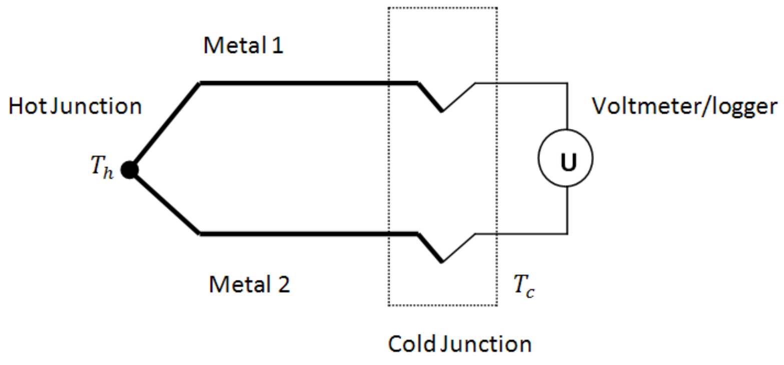


Figure 6.4: Thermocouple measurement curcuit.

6.3 Experiments

Many possible scenarios were simulated before the experiments. The goal was to find an optimal set up with respect to fire size, HRR, cross-section pipe dimensions and the height of the pipes above the fire within limits of measurement range of equipments and other laboratory facilities. Another goal was to find in which combinations of the varied parameters that gave largest differences when comparing a fire plume with and without obstacles. Circular pipes, which were used in Skarsbøs experiments, were replaced with 60 mm x 60 mm (outer dimensions and thickness of 4 mm) square pipes. Unfortunately, only round-edged pipes were available in the area of Haugesund at moment the experiments were performed. However, a somewhat smaller dimension was preferable for a optimal turbulence intensity. But since a smaller dimension requires a finer resolution of the grid in simulations to capture the correct fluid flow between the pipes, 60 mm x 60 mm was an acceptable compromise.

Simulations showed that the pipes placed in the persistent flame region where the flow is accelerating (see Figure 2.7) gave highest rise in temperature and velocity. This is about equivalent to heights below 1/3 of the flame height. Furthermore, a distance of 60 mm between the pipes was used in the simulations. This distance was later changed to 40 mm after some test experiments revealed that the flow around and just above the obstacles got more stable.

In Table 6.1 the planned scenarios are given. The experiment set up is seen in Figure 6.5.

Table 6.1: Experimental scenarios.

Number	Fuel [-]	HRR [kW/m^2]	Heights of pipes [m]
1.1	Propane	500	0.15
1.2	Propane	500	0.30
1.3	Propane	500	0.45
2.1	Propane	1000	0.15
2.2	Propane	1000	0.30
2.3	Propane	1000	0.45
3.1	Propane	1500	0.15
3.2	Propane	1500	0.30
3.3	Propane	1500	0.45
4.1	Heptane	-	0.15
4.2	Heptane	-	0.30
4.3	Heptane	-	0.45

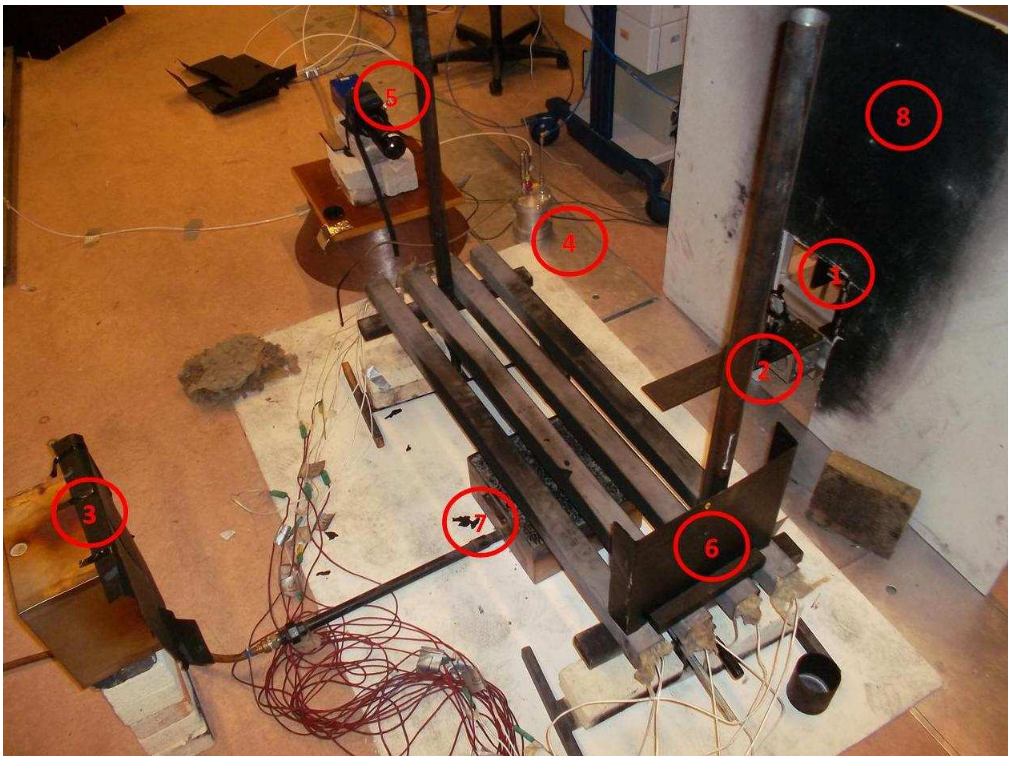


Figure 6.5: Experimental setup: (1) laser, (2) lens, (3) laser beam absorber, (4) seeding box, (5) camera, (6) anti-reflection shield, (7) gas burner and (8) wind shield.

6.3.1 Summary of the experiments

The laboratory that was booked for the experiments was not suited for fire experiments. It was used for enclosed bench scale flame experiments to study combustion processes in detail. Only four days were spent in the laboratory which turned out to be way too little time. Below, some limitations because of this and challenges during the experiments are described.

Scenarios

Initially, a series of 9 experiments was planned with a propane fire with HRR of 500 kW/m^2, 1000 kW/m^2 and 1500 kW/m^2, with pipes in heights of 0.15 m, 0.3 m and 0.45 m. Because the laboratory was loaded with a lot of expensive equipment and quite small relatively to the fire size, the plan was not achievable. Only experiments 500 kW/m^2 propane fire and 32 cm x 32 cm liquid methanol was performed.

Smoke hood and ventilation

The relative large fire size that was initially planned made it necessary to enlarged the smoke hood and set the ventilation on full power to prevent spread of smoke and hazardous seeding particles in the room. Another problem was then observed, the small fire got unstable because of the momentum from the ventilation and tilted in away from the laser. The problem was solved by placing a wind shield (number 8 in Figure 6.5) in front of the laser. However, this problem was not observed for larger fire (>500 kW/m^2).

Calibration of the Laser and Camera

Calibration of laser and camera is the most time consuming part of the experiments. First, the camera must be focused on the measurement area, as seen in Figure 6.6. In these calibration experiments, the measurement area was set to 4 cm x 4 cm between the two pipes on the left side. Large measurement areas requires a strong laser to penetrate flame, especially for sooty flames. One of the uncertainties before doing the experiments was whether the laser used could penetrate such a sooty flame produced by propane. That is why methanol was chosen in the calibration experiments.

The camera does not necessarily has to be placed perpendicular to the laser sheet, since the distance between the dots on the calibration plate is known (in Figure 6.6). The correction was automatically fixed by the logging software that was used.

Since the seeding particles are traced as they are illuminated by the laser pulses, disturbing incoming light on the camera lens must be minimized. For this reason a black anti-reflection shield (number (6) in Figure 6.5) was placed behind the measurement area to absorb light in all wave lengths. Another uncertainty that arose regarding the disturbing light was the use of propane as fuel. Propane flames are sootier than for example methanol and are also therefore also emitting more thermal radiation.

The laser used was only able to operate on 4 Hz. This is not sufficient to capture turbulent motions. So the plan about studying turbulent characteristics was discarded in the early stages. During the calibration, the shutter on the camera did only function sporadically. As a result, too much light was received on every second image making the tracing of particles impossible. From the experiments no conclusion could be draw whether a propane flame is too sooty to measure a velocity vector field with the equipment available at Lund University.

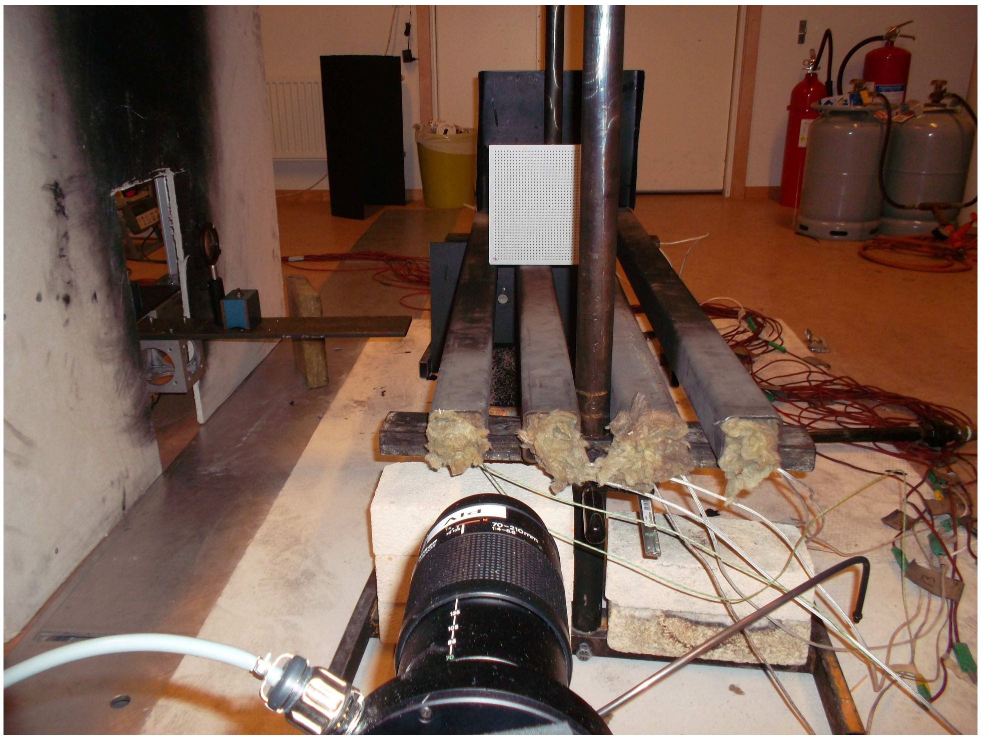


Figure 6.6: Calibration of the camera.

Seeding

Seeding of particles is the most challenging part of PIV measurements of flames. First of all, no universal method for seeding particles in flames exists. Secondly, the handling of seeding particles requires uttermost care due to hazards of inhaling it. Breathing masks and powerful ventilation are demanded. During these experiments, the flame was seeded in three different ways; by smoke sticks, smoke pellets and seeding of particles with pressurized air. All methods were some way unsuitable and the out coming results were of rather poor quality, primarily caused by the methods. Seeding with smoke was unsuitable because a stable concentration of smoke particles within measurable range was not achievable in the whole measurement area on a sufficient number of the images recorded. It was also attempted to seed the particles by pressurized in vertical direction, both upward and downward, and also horizontal direction faced in positive direction along the laser beam. When the particles were seeded upwards, a jet was clearly observed in the results. It was obvious that the particles were strongly influenced by the release momentum and did not move free with the local flow velocity in the flame. In downward direction, seeding particles got stuck in burner. The measurement area was not provided with enough seeding when the seeding particles were supplied horizontally. Furthermore, the flame behavior was too much influenced when pressurized air method was applied for small fires.

Chapter 7

Conclusion

The PIV data from the experiments at Lund University were of rather poor quality and unsuitable for CFD validation. Too much light was received in the camera lens since the shutter on the camera only functioned sporadically on every second image. The seeding of particles turned out to be quite challenging and no successful method was found. Smoke pellets and smoke sticks did not provide enough seeding within measurable range in the desired area. The seeding particles did not move free with the local flow velocity when they were seeded by pressurized air, but were influenced by the release momentum. The flame behavior for small fires was also influenced by the pressurized air method

The validation of FDS-EDC revealed that the model is quite grid dependent. Except from the grid dependency the implemented code is in satisfactory agreement with the validation cases, and as good as the already existing or even better. Both models are about the same computational expensive. However, a single model constant, C_{LES}, gives not the most correct result for all the cases. An overall recommended C_{LES} of 0.015, as well as the already existing combustion model, over estimates the flame height for $Q^* \lesssim 1$. This over estimation of flame height is not observed with the Smagorinsky turbulence model. Furthermore, the maximum temperature is as much as 600°C with the Deardorff model for the McCaffery simulation of 57 kW. In addition, the temperature peak was located too far above the burner. The Smagorinsky model did not displace this peak in such extent as Deardorff and Vreman. For the McCaffery simulations, all turbulence models gave results in better agreement with $C_{LES} = 0.01$ than $C_{LES} = 0.015$. The Smagorinsky model, was the only turbulence model that captured the dip for the vertical velocity in the Sandia experiment test 17, but with the finest grid the CPU clock time was nearly 70% larger than the two other turbulence models.

Before LES-EDC can be applied in fire analysis the model must be modified to be grid independent. The model should also be be validated in a wider range for temperature and velocity profiles in practical fire scale to establish a more accurate constant. A dynamic constant is not necessary for buoyancy-driven fires in the context of fire engineering, but could a benefit for detailed investigation of flames.

7.1 Further Work

The author of this thesis and the participants of the experiments at Lund University discussed alternative ways to arrange experiments to achieve successful PIV measurements. Regarding the seeding, the most challenging part;

- the first step is to perform the experiments i an closed room where seeding particles are in no hazard to humans. Then it is possible to seed the flame with large amounts of particles.
- step two is to customize the burner and build in seeding point around the frame (maybe even in the middle of the burner too) and seed the particles with low-velocity controllable pressurized air.

The implemented code is validated against flame height i range $0.1 \leq Q^* \leq 10000$; radial and vertical (horizontal) velocity profiles in range $1.35 \leq Q^* \leq 2.2$; centerline velocity and temperature profiles in range $0.005 \leq Q^* \leq 0.02$. In further work, it could be interesting to investigate how the mentioned parameter interact with each other when C_{LES} is varied. The code should also be validated for temperature and velocity in the range of $0.1 \lesssim Q^* \lesssim 1.5$ where the FDS-EDC did not fit well for the flame height (except from Smagorinsky turbulence model). The flame height is varying significantly because the unstable nature of fire [38]. This puff cycle could also be validated in further work [34]. At last the model has to be modified to be grid independent.

Bibliography

[1] B.F.Magnussen. On the structure of turbulence and a generalized eddy dissipation concept for chemical reaction in turbulent flow. In *American Institute of Aeronautics and Astronautics, Aerospace Science Meeting*, number 19th, St. Louis, MusSouri, USA, January 1981.

[2] B.F. Magnussen and I.S. Ertesvåg. The eddy dissipation turbulence energy cascade model. *Combustion Science and Technology*, 159(1):213–235, Desember 2000.

[3] B.F.Magnussen. The eddy dissipation concept a bridge between science and technology. In *ECCOMAS Thematic Conference on Computational Combustion*, Lisbon, June 2005.

[4] K. McGrattan; R. McDermott; S. Hostikka and J. Floyd. *Fire Dynamics Fire Simulator (Version 5) User's Guide.* NIST, Washington, 2010.

[5] K.E. Rian B. Panjwani, I.S.Ertesvåg and A. Gruber. Subgrid combution modelling for large eddy simulations (les) of turbulent combustion using eddy dissipation concept (edc). pages 1–19, Lisbon, June 2010.

[6] B. Panjwani. *Large Eddy Simulation of Turbulent Combustion With Chemical Kinetics:The Eddy Dissipation Concept as a Subgrid Combustion Model for Large Eddy Simulation: Theory and Validation.* PhD thesis, Norwegian University of Science and Technology, Trondheim, 2011.

[7] L.X. Zhou L.Y. Hu and J.Zhang. Large-eddy simulation of a swirling diffusion flame using a som sgs combustion model. *Numerical Heat Transfer, Part B: Fundamentals: An International Journal of Computation and Methodology*, 50(1):41–58, 2006.

[8] L.Y Hu L.X. Zhou and F. Wang. Large-eddy simulation of turbulent combustion using som and ebu sgs combustion models. pages 99–102, 2009.

[9] J. D. Anderson Jr. *Computational Fluid Dynamics.* Number pp. 1-167. McGRAW-HILL, New York, 1th edition, 1995.

[10] B. Karlsson and J.G. Quintiere. *Enclosure Fire Dynamics.* CRC press, Boca Raton, 1st edition, 1999.

[11] S.R. Turns. *An Introduction tho Combustion, Concepts and Applications.* McGraw-Hill, Boston, 2nd edition, 1999.

[12] H. Tennekes and J.L. Lumley. *A First Course in Turbulence.* The MIT Press, Cambridge, 1972.

[13] I.S. Ertesvåg. *Turbulent Strøyming og Forbrenning.* Tapir, Trondheim, 1st edition, 2000.

[14] J. Warnatz; U. Mass and R.W. Dibble. *Combustion.* Number pp. 185-212. Springer, Germany, 4th edition, 2006.

[15] G.H. Yeoh and K.K. Yuen. *Computational Fluid Dynamics in Fire Engineering.* Elsevier, Amsterdam, 1st edition, 2009.

[16] B.E Launder and D.B Spalding. The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 3(2):269–289, August 1974.

[17] M. Germano; U. Piomelli; P. Moin and W. Cabot. A dynamic subgrid-scale eddy viscosity model. *Phys. Fluids A*, 3(7):1760–1765, November 1991.

[18] Dennis Veynante Thierry Poinsot. *Theoretical and Numerical Combustion.* Edwards, Philadelphia, 2nd edition, 2005.

[19] A. W. Vreman. An eddy-viscosity subgrid-scale model for turbulent shear flow: Algebraic theory and applications. *Phys. Fluids*, 16(10):3670–3681, September 2004.

[20] D. Drysdale. *An Introduction to Fire Dynamics.* Number pp. 1-29. John Wiley and Sons, Chichester, 2nd edition, 2007.

[21] J. Chen S. McAllister and A.C. Fernandesz-Pello. *Fundamentals of Combustion Processes.* Springer, April 2011.

[22] R. Borghi and M. Destriau. *Combustion and Flames, Chemical and Physical Principles.* Éditions Technip, Paris, 1998.

[23] G. Heskestad. *SFPE Handbook of Fire Protection Engineering, chapter Fire Plumes, Flame Height and Air Entrainment.* National Fire Protection Association (NFPA), 3rd edition, 2002.

[24] B.J. MacCaffery. Purely boyant diffusion flames: Some experimental results. (NBSIR 79-1910), October 1979.

[25] K. McGrattan; R. McDermott; S. Hostikka and J. Floyd. *Fire Dynamics Fire Simulator Technical Reference Guide, Volume 1: Mathematical Model.* NIST, Washington.

[26] J.E. Floyd. *Multi-Parameter, Multiple Fuel Mixture Combustion Model for Fire Dynamics Simulator.* NIST, Baltimore, November 2008.

[27] B. Husted J. Carlsson and U. Göransson. Fordele og faldgruber ved anvendelse af cfd til brandtekniske beregninger. *HVAC magasinet*, pages 44–47, July 2005.

[28] E. Gissi. *An Introduction to Fire Simulation with FDS and Smokeview*, September 2009.

[29] T. Yamamoto H. Aoki M. Yaga, H. Endo and T. Mirua. Modeling of eddy characteristic time in les for calculating tubrulent diffusion time. *International Journal of Heat and Mass Transfer*, 45(11):2343–2349, 2002.

[30] American Society for Testing and Materials, West Conshohocken, West Conshohocken, Pennsylvania. ASTM E 1355-04. *Standard Guide for Evaluating the Predictive Capabilities of Deterministic Fire Models*, 1 edition, 2004. i.

[31] K. McGrattan; R. McDermott; S. Hostikka and J. Floyd. *Fire Dynamics Simulator (Version 5) Technical Reference Guide, Volume 2: Verification.* NIST, Washington, 2010.

[32] K. McGrattan; R. McDermott; S. Hostikka and J. Floyd. *Fire Dynamics Fire Simulator Technical Reference Guide, Volume 3: Validation.* NIST, Washington, 2010.

[33] E. J. Weckman S. R. Tieszen, T. J. O'Hern and R. W. Schefer. Experimental study of the effect of fuel mass flux on a 1-m-diameter methane fire and comparison with a hydrogen fire. *Combustion and Flame*, 139(126-141), 2004.

[34] R. W. Schefer E. J. Weckman S. R. Tieszen, T. J. O'Hern and T. K. Blanchat. Experimental study of the flow field in and around a one meter diameter methane fire. *Combustion and Flame*, 129:378–391, 2002.

[35] Lars Roar Skarsbo. *An Experimental Study of Pool Fire and Validation of Different CFD Models.* PhD thesis, University of Bergen, June 2011.

[36] S.T Wereley M. Raffel, C.E Willert and J. Kompenhans. *Particle Image Velocimetry: A Practical Guide.* Springer, Berlin, 2nd edition, 1998.

[37] Particle image velocimetry (piv); `www.dantecdynamics.com`, March 2012.

[38] Y. Xin; J.P. Gore; K.B. McGrattan; R.G. Rehm and H.R. Baum. Fire dynamics of a turbulent buoyant flame using mixture-fraction-based combustion model. *Combustion and Flame*, 141:329–335, 2005.

Appendix A

FDS input files

A.1 Sandia test 14

```
&HEAD CHID='Sandia_CH4_1m_Test14_dx1p5cm', TITLE='Sandia 1m low
     flow rate methane pool fire (Test 14), 1.5 cm resolution' /

&MULT ID='mesh array', DX=1.5,DY=1.5,DZ=1.0, I_UPPER=1,J_UPPER
    =1,K_UPPER=3 /
&MESH IJK=96,96,64, XB=-1.5,0.0,-1.5,0.0,-.0625,0.9375, MULT_ID
    ='mesh array' /

&TIME T_END=20. /
&MISC TMPA=12.
      P_INF=80600.
      TURBULENCE_MODEL='VREMAN' /

&REAC FUEL='METHANE'
      HEAT_OF_COMBUSTION=50611.
      CRITICAL_FLAME_TEMPERATURE=1207 /

&RADI RADIATIVE_FRACTION=0.2 /
&DUMP SIG_FIGS=4, SIG_FIGS_EXP=2 /
&MATL ID='STEEL', CONDUCTIVITY=54., SPECIFIC_HEAT=0.465,
    DENSITY=7850., EMISSIVITY=0.9 /
&SURF ID='PLATE', COLOR='GRAY', DEFAULT=.TRUE., MATL_ID='STEEL
    ', THICKNESS=0.025 /

&SURF ID='POOL',MASS_FLUX(1)=0.04,SPEC_ID(1)='METHANE',
    TMP_FRONT=11. /
```

```
&VENT XB=-0.5,0.5,-0.5,0.5,-.0625,-.0625,COLOR='BLUE',SURF_ID='
    POOL'/

&VENT XB=-1.5,-.5,-1.5,1.5,-.0625,-.0625,SURF_ID='OPEN'/
&VENT XB=.5,1.5,-1.5,1.5,-.0625,-.0625,SURF_ID='OPEN'/
&VENT XB=-1.5,1.5,-1.5,-.5,-.0625,-.0625,SURF_ID='OPEN'/
&VENT XB=-1.5,1.5,.5,1.5,-.0625,-.0625,SURF_ID='OPEN'/

&VENT MB='ZMAX',SURF_ID='OPEN'/
&VENT MB='YMIN',SURF_ID='OPEN'/
&VENT MB='YMAX',SURF_ID='OPEN'/
&VENT MB='XMIN',SURF_ID='OPEN'/
&VENT MB='XMAX',SURF_ID='OPEN'/

&SLCF PBY=-.0625, QUANTITY='VELOCITY', VECTOR=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='DENSITY', CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='TEMPERATURE', CELL_CENTERED=.TRUE.
    /
&SLCF PBY=-.0625, QUANTITY='HRRPUV', CELL_CENTERED=.TRUE. /
/&SLCF PBY=-.0625, QUANTITY='HRRPUA', CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='MASS FRACTION', SPEC_ID='METHANE',
    CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='C_SMAG', CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='TURBULENCE RESOLUTION',
    CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='WAVELET ERROR', QUANTITY2='MASS
    FRACTION', SPEC_ID='METHANE', CELL_CENTERED=.TRUE. /

&OBST XB=-0.70711,0.70711,-0.70711,0.70711,-0.0625,0 /
&OBST XB=-0.70711,0.70711,-0.70711,0.70711,-0.0625,0 /
&OBST XB=-0.69211,0.69211,-0.7218,0.7218,-0.0625,0 /
&OBST XB=-0.7218,0.7218,-0.69211,0.69211,-0.0625,0 /
&OBST XB=-0.67711,0.67711,-0.73588,0.73588,-0.0625,0 /
&OBST XB=-0.73588,0.73588,-0.67711,0.67711,-0.0625,0 /
&OBST XB=-0.66211,0.66211,-0.74941,0.74941,-0.0625,0 /
&OBST XB=-0.74941,0.74941,-0.66211,0.66211,-0.0625,0 /
&OBST XB=-0.64711,0.64711,-0.7624,0.7624,-0.0625,0 /
&OBST XB=-0.7624,0.7624,-0.64711,0.64711,-0.0625,0 /
&OBST XB=-0.63211,0.63211,-0.77488,0.77488,-0.0625,0 /
&OBST XB=-0.77488,0.77488,-0.63211,0.63211,-0.0625,0 /
&OBST XB=-0.61711,0.61711,-0.78688,0.78688,-0.0625,0 /
&OBST XB=-0.78688,0.78688,-0.61711,0.61711,-0.0625,0 /
&OBST XB=-0.60211,0.60211,-0.79842,0.79842,-0.0625,0 /
```

```
&OBST XB=−0.79842 ,0.79842 , −0.60211 ,0.60211 , −0.0625 ,0 /
&OBST XB=−0.58711 ,0.58711 , −0.80951 ,0.80951 , −0.0625 ,0 /
&OBST XB=−0.80951 ,0.80951 , −0.58711 ,0.58711 , −0.0625 ,0 /
&OBST XB=−0.57211 ,0.57211 , −0.82018 ,0.82018 , −0.0625 ,0 /
&OBST XB=−0.82018 ,0.82018 , −0.57211 ,0.57211 , −0.0625 ,0 /
&OBST XB=−0.55711 ,0.55711 , −0.83044 ,0.83044 , −0.0625 ,0 /
&OBST XB=−0.83044 ,0.83044 , −0.55711 ,0.55711 , −0.0625 ,0 /
&OBST XB=−0.54211 ,0.54211 , −0.84031 ,0.84031 , −0.0625 ,0 /
&OBST XB=−0.84031 ,0.84031 , −0.54211 ,0.54211 , −0.0625 ,0 /
&OBST XB=−0.52711 ,0.52711 , −0.8498 ,0.8498 , −0.0625 ,0 /
&OBST XB=−0.8498 ,0.8498 , −0.52711 ,0.52711 , −0.0625 ,0 /
&OBST XB=−0.51211 ,0.51211 , −0.85892 ,0.85892 , −0.0625 ,0 /
&OBST XB=−0.85892 ,0.85892 , −0.51211 ,0.51211 , −0.0625 ,0 /
&OBST XB=−0.49711 ,0.49711 , −0.86769 ,0.86769 , −0.0625 ,0 /
&OBST XB=−0.86769 ,0.86769 , −0.49711 ,0.49711 , −0.0625 ,0 /
&OBST XB=−0.48211 ,0.48211 , −0.87611 ,0.87611 , −0.0625 ,0 /
&OBST XB=−0.87611 ,0.87611 , −0.48211 ,0.48211 , −0.0625 ,0 /
&OBST XB=−0.46711 ,0.46711 , −0.8842 ,0.8842 , −0.0625 ,0 /
&OBST XB=−0.8842 ,0.8842 , −0.46711 ,0.46711 , −0.0625 ,0 /
&OBST XB=−0.45211 ,0.45211 , −0.89196 ,0.89196 , −0.0625 ,0 /
&OBST XB=−0.89196 ,0.89196 , −0.45211 ,0.45211 , −0.0625 ,0 /
&OBST XB=−0.43711 ,0.43711 , −0.89941 ,0.89941 , −0.0625 ,0 /
&OBST XB=−0.89941 ,0.89941 , −0.43711 ,0.43711 , −0.0625 ,0 /
&OBST XB=−0.42211 ,0.42211 , −0.90655 ,0.90655 , −0.0625 ,0 /
&OBST XB=−0.90655 ,0.90655 , −0.42211 ,0.42211 , −0.0625 ,0 /
&OBST XB=−0.40711 ,0.40711 , −0.91338 ,0.91338 , −0.0625 ,0 /
&OBST XB=−0.91338 ,0.91338 , −0.40711 ,0.40711 , −0.0625 ,0 /
&OBST XB=−0.39211 ,0.39211 , −0.91992 ,0.91992 , −0.0625 ,0 /
&OBST XB=−0.91992 ,0.91992 , −0.39211 ,0.39211 , −0.0625 ,0 /
&OBST XB=−0.37711 ,0.37711 , −0.92617 ,0.92617 , −0.0625 ,0 /
&OBST XB=−0.92617 ,0.92617 , −0.37711 ,0.37711 , −0.0625 ,0 /
&OBST XB=−0.36211 ,0.36211 , −0.93214 ,0.93214 , −0.0625 ,0 /
&OBST XB=−0.93214 ,0.93214 , −0.36211 ,0.36211 , −0.0625 ,0 /
&OBST XB=−0.34711 ,0.34711 , −0.93783 ,0.93783 , −0.0625 ,0 /
&OBST XB=−0.93783 ,0.93783 , −0.34711 ,0.34711 , −0.0625 ,0 /
&OBST XB=−0.33211 ,0.33211 , −0.94324 ,0.94324 , −0.0625 ,0 /
&OBST XB=−0.94324 ,0.94324 , −0.33211 ,0.33211 , −0.0625 ,0 /
&OBST XB=−0.31711 ,0.31711 , −0.94839 ,0.94839 , −0.0625 ,0 /
&OBST XB=−0.94839 ,0.94839 , −0.31711 ,0.31711 , −0.0625 ,0 /
&OBST XB=−0.30211 ,0.30211 , −0.95327 ,0.95327 , −0.0625 ,0 /
&OBST XB=−0.95327 ,0.95327 , −0.30211 ,0.30211 , −0.0625 ,0 /
&OBST XB=−0.28711 ,0.28711 , −0.9579 ,0.9579 , −0.0625 ,0 /
&OBST XB=−0.9579 ,0.9579 , −0.28711 ,0.28711 , −0.0625 ,0 /
```

```
&OBST XB=−0.27211,0.27211,−0.96227,0.96227,−0.0625,0 /
&OBST XB=−0.96227,0.96227,−0.27211,0.27211,−0.0625,0 /
&OBST XB=−0.25711,0.25711,−0.96638,0.96638,−0.0625,0 /
&OBST XB=−0.96638,0.96638,−0.25711,0.25711,−0.0625,0 /
&OBST XB=−0.24211,0.24211,−0.97025,0.97025,−0.0625,0 /
&OBST XB=−0.97025,0.97025,−0.24211,0.24211,−0.0625,0 /
&OBST XB=−0.22711,0.22711,−0.97387,0.97387,−0.0625,0 /
&OBST XB=−0.97387,0.97387,−0.22711,0.22711,−0.0625,0 /
&OBST XB=−0.21211,0.21211,−0.97725,0.97725,−0.0625,0 /
&OBST XB=−0.97725,0.97725,−0.21211,0.21211,−0.0625,0 /
&OBST XB=−0.19711,0.19711,−0.98038,0.98038,−0.0625,0 /
&OBST XB=−0.98038,0.98038,−0.19711,0.19711,−0.0625,0 /
&OBST XB=−0.18211,0.18211,−0.98328,0.98328,−0.0625,0 /
&OBST XB=−0.98328,0.98328,−0.18211,0.18211,−0.0625,0 /
&OBST XB=−0.16711,0.16711,−0.98594,0.98594,−0.0625,0 /
&OBST XB=−0.98594,0.98594,−0.16711,0.16711,−0.0625,0 /
&OBST XB=−0.15211,0.15211,−0.98836,0.98836,−0.0625,0 /
&OBST XB=−0.98836,0.98836,−0.15211,0.15211,−0.0625,0 /
&OBST XB=−0.13711,0.13711,−0.99056,0.99056,−0.0625,0 /
&OBST XB=−0.99056,0.99056,−0.13711,0.13711,−0.0625,0 /
&OBST XB=−0.12211,0.12211,−0.99252,0.99252,−0.0625,0 /
&OBST XB=−0.99252,0.99252,−0.12211,0.12211,−0.0625,0 /
&OBST XB=−0.10711,0.10711,−0.99425,0.99425,−0.0625,0 /
&OBST XB=−0.99425,0.99425,−0.10711,0.10711,−0.0625,0 /
&OBST XB=−0.092107,0.092107,−0.99575,0.99575,−0.0625,0 /
&OBST XB=−0.99575,0.99575,−0.092107,0.092107,−0.0625,0 /
&OBST XB=−0.077107,0.077107,−0.99702,0.99702,−0.0625,0 /
&OBST XB=−0.99702,0.99702,−0.077107,0.077107,−0.0625,0 /
&OBST XB=−0.062107,0.062107,−0.99807,0.99807,−0.0625,0 /
&OBST XB=−0.99807,0.99807,−0.062107,0.062107,−0.0625,0 /
&OBST XB=−0.047107,0.047107,−0.99889,0.99889,−0.0625,0 /
&OBST XB=−0.99889,0.99889,−0.047107,0.047107,−0.0625,0 /
&OBST XB=−0.032107,0.032107,−0.99948,0.99948,−0.0625,0 /
&OBST XB=−0.99948,0.99948,−0.032107,0.032107,−0.0625,0 /
&OBST XB=−0.017107,0.017107,−0.99985,0.99985,−0.0625,0 /
&OBST XB=−0.99985,0.99985,−0.017107,0.017107,−0.0625,0 /
&OBST XB=−0.0021068,0.0021068,−1,1,−0.0625,0 /
&OBST XB=−1,1,−0.0021068,0.0021068,−0.0625,0 /

&HOLE XB=−0.35355,0.35355,−0.35355,0.35355,−.125,.1/
&HOLE XB=−0.35355,0.35355,−0.35355,0.35355,−.125,.1/
&HOLE XB=−0.33755,0.33755,−0.36886,0.36886,−.125,.1/
&HOLE XB=−0.36886,0.36886,−0.33755,0.33755,−.125,.1/
```

```
&HOLE XB=-0.32155,0.32155,-0.38289,0.38289,-.125,.1/
&HOLE XB=-0.38289,0.38289,-0.32155,0.32155,-.125,.1/
&HOLE XB=-0.30555,0.30555,-0.39577,0.39577,-.125,.1/
&HOLE XB=-0.39577,0.39577,-0.30555,0.30555,-.125,.1/
&HOLE XB=-0.28955,0.28955,-0.40763,0.40763,-.125,.1/
&HOLE XB=-0.40763,0.40763,-0.28955,0.28955,-.125,.1/
&HOLE XB=-0.27355,0.27355,-0.41853,0.41853,-.125,.1/
&HOLE XB=-0.41853,0.41853,-0.27355,0.27355,-.125,.1/
&HOLE XB=-0.25755,0.25755,-0.42856,0.42856,-.125,.1/
&HOLE XB=-0.42856,0.42856,-0.25755,0.25755,-.125,.1/
&HOLE XB=-0.24155,0.24155,-0.43778,0.43778,-.125,.1/
&HOLE XB=-0.43778,0.43778,-0.24155,0.24155,-.125,.1/
&HOLE XB=-0.22555,0.22555,-0.44623,0.44623,-.125,.1/
&HOLE XB=-0.44623,0.44623,-0.22555,0.22555,-.125,.1/
&HOLE XB=-0.20955,0.20955,-0.45397,0.45397,-.125,.1/
&HOLE XB=-0.45397,0.45397,-0.20955,0.20955,-.125,.1/
&HOLE XB=-0.19355,0.19355,-0.46102,0.46102,-.125,.1/
&HOLE XB=-0.46102,0.46102,-0.19355,0.19355,-.125,.1/
&HOLE XB=-0.17755,0.17755,-0.46741,0.46741,-.125,.1/
&HOLE XB=-0.46741,0.46741,-0.17755,0.17755,-.125,.1/
&HOLE XB=-0.16155,0.16155,-0.47318,0.47318,-.125,.1/
&HOLE XB=-0.47318,0.47318,-0.16155,0.16155,-.125,.1/
&HOLE XB=-0.14555,0.14555,-0.47835,0.47835,-.125,.1/
&HOLE XB=-0.47835,0.47835,-0.14555,0.14555,-.125,.1/
&HOLE XB=-0.12955,0.12955,-0.48292,0.48292,-.125,.1/
&HOLE XB=-0.48292,0.48292,-0.12955,0.12955,-.125,.1/
&HOLE XB=-0.11355,0.11355,-0.48693,0.48693,-.125,.1/
&HOLE XB=-0.48693,0.48693,-0.11355,0.11355,-.125,.1/
&HOLE XB=-0.097553,0.097553,-0.49039,0.49039,-.125,.1/
&HOLE XB=-0.49039,0.49039,-0.097553,0.097553,-.125,.1/
&HOLE XB=-0.081553,0.081553,-0.4933,0.4933,-.125,.1/
&HOLE XB=-0.4933,0.4933,-0.081553,0.081553,-.125,.1/
&HOLE XB=-0.065553,0.065553,-0.49568,0.49568,-.125,.1/
&HOLE XB=-0.49568,0.49568,-0.065553,0.065553,-.125,.1/
&HOLE XB=-0.049553,0.049553,-0.49754,0.49754,-.125,.1/
&HOLE XB=-0.49754,0.49754,-0.049553,0.049553,-.125,.1/
&HOLE XB=-0.033553,0.033553,-0.49887,0.49887,-.125,.1/
&HOLE XB=-0.49887,0.49887,-0.033553,0.033553,-.125,.1/
&HOLE XB=-0.017553,0.017553,-0.49969,0.49969,-.125,.1/
&HOLE XB=-0.49969,0.49969,-0.017553,0.017553,-.125,.1/
&HOLE XB=-0.0015534,0.0015534,-0.5,0.5,-.125,.1/
&HOLE XB=-0.5,0.5,-0.0015534,0.0015534,-.125,.1/
```

```
Radial profiles of velocity and mass fraction

&DEVC XB=-0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='W-VELOCITY', ID
    ='Wp3', POINTS=21, X_ID='x' /
&DEVC XB=-0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='W-VELOCITY', ID
    ='Wp5', POINTS=21, HIDE_COORDINATES=.TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='W-VELOCITY', ID
    ='Wp9', POINTS=21, HIDE_COORDINATES=.TRUE. /

&DEVC XB=-0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='U-VELOCITY', ID
    ='Up3', POINTS=21, HIDE_COORDINATES=.TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='U-VELOCITY', ID
    ='Up5', POINTS=21, HIDE_COORDINATES=.TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='U-VELOCITY', ID
    ='Up9', POINTS=21, HIDE_COORDINATES=.TRUE. /

&DEVC XB=-0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='W-VELOCITY', ID
    ='Wp3_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='W-VELOCITY', ID
    ='Wp5_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='W-VELOCITY', ID
    ='Wp9_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /

&DEVC XB=-0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='U-VELOCITY', ID
    ='Up3_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='U-VELOCITY', ID
    ='Up5_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='U-VELOCITY', ID
    ='Up9_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /

&TAIL /
```

A.2 Sandia test 17

```
&HEAD CHID='Sandia_CH4_1m_Test17_dx1p5cm', TITLE='Sandia 1m
    high flow rate methane pool fire (Test 17), 1.5 cm
    resolution' /
```

```
&MULT ID='mesh array', DX=1.5,DY=1.5,DZ=1.0, I_UPPER=1,J_UPPER
    =1,K_UPPER=3 /
&MESH IJK=96,96,64, XB=-1.5,0.0,-1.5,0.0,-.0625,0.9375, MULT_ID
    ='mesh array' /

&TIME T_END=20. /
&MISC TMPA=5.
       P_INF=81100.
       TURBULENCE_MODEL='VREMAN'/

&REAC FUEL='METHANE'
       HEAT_OF_COMBUSTION=50350.
       CRITICAL_FLAME_TEMPERATURE=1207 /

&RADI RADIATIVE_FRACTION=0.2 /
&DUMP SIG_FIGS=4, SIG_FIGS_EXP=2 /
&MATL ID='STEEL', CONDUCTIVITY=54., SPECIFIC_HEAT=0.465,
    DENSITY=7850., EMISSIVITY=0.9 /
&SURF ID='PLATE', COLOR='GRAY', DEFAULT=.TRUE., MATL_ID='STEEL
    ', THICKNESS=0.025 /

&SURF ID='POOL',MASS_FLUX(1)=0.066,SPEC_ID(1)='METHANE',
    TMP_FRONT=1. /

&VENT XB=-0.5,0.5,-0.5,0.5,-.0625,-.0625,COLOR='BLUE',SURF_ID='
    POOL'/

&VENT XB=-1.5,-.5,-1.5,1.5,-.0625,-.0625,SURF_ID='OPEN'/
&VENT XB=.5,1.5,-1.5,1.5,-.0625,-.0625,SURF_ID='OPEN'/
&VENT XB=-1.5,1.5,-1.5,-.5,-.0625,-.0625,SURF_ID='OPEN'/
&VENT XB=-1.5,1.5,.5,1.5,-.0625,-.0625,SURF_ID='OPEN'/

&VENT MB='ZMAX',SURF_ID='OPEN'/
&VENT MB='YMIN',SURF_ID='OPEN'/
&VENT MB='YMAX',SURF_ID='OPEN'/
&VENT MB='XMIN',SURF_ID='OPEN'/
&VENT MB='XMAX',SURF_ID='OPEN'/

&SLCF PBY=-.0625, QUANTITY='VELOCITY', VECTOR=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='DENSITY', CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='TEMPERATURE', CELL_CENTERED=.TRUE.
    /
```

```
&SLCF PBY=-.0625, QUANTITY='HRRPUV', CELL_CENTERED=.TRUE. /
/&SLCF PBY=-.0625, QUANTITY='HRRPUA', CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='MASS FRACTION', SPEC_ID='METHANE',
   CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='C_SMAG', CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='TURBULENCE RESOLUTION',
   CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='WAVELET ERROR', QUANTITY2='MASS
   FRACTION', SPEC_ID='METHANE', CELL_CENTERED=.TRUE. /

&OBST XB=-0.70711,0.70711,-0.70711,0.70711,-0.0625,0 /
&OBST XB=-0.70711,0.70711,-0.70711,0.70711,-0.0625,0 /
&OBST XB=-0.69211,0.69211,-0.7218,0.7218,-0.0625,0 /
&OBST XB=-0.7218,0.7218,-0.69211,0.69211,-0.0625,0 /
&OBST XB=-0.67711,0.67711,-0.73588,0.73588,-0.0625,0 /
&OBST XB=-0.73588,0.73588,-0.67711,0.67711,-0.0625,0 /
&OBST XB=-0.66211,0.66211,-0.74941,0.74941,-0.0625,0 /
&OBST XB=-0.74941,0.74941,-0.66211,0.66211,-0.0625,0 /
&OBST XB=-0.64711,0.64711,-0.7624,0.7624,-0.0625,0 /
&OBST XB=-0.7624,0.7624,-0.64711,0.64711,-0.0625,0 /
&OBST XB=-0.63211,0.63211,-0.77488,0.77488,-0.0625,0 /
&OBST XB=-0.77488,0.77488,-0.63211,0.63211,-0.0625,0 /
&OBST XB=-0.61711,0.61711,-0.78688,0.78688,-0.0625,0 /
&OBST XB=-0.78688,0.78688,-0.61711,0.61711,-0.0625,0 /
&OBST XB=-0.60211,0.60211,-0.79842,0.79842,-0.0625,0 /
&OBST XB=-0.79842,0.79842,-0.60211,0.60211,-0.0625,0 /
&OBST XB=-0.58711,0.58711,-0.80951,0.80951,-0.0625,0 /
&OBST XB=-0.80951,0.80951,-0.58711,0.58711,-0.0625,0 /
&OBST XB=-0.57211,0.57211,-0.82018,0.82018,-0.0625,0 /
&OBST XB=-0.82018,0.82018,-0.57211,0.57211,-0.0625,0 /
&OBST XB=-0.55711,0.55711,-0.83044,0.83044,-0.0625,0 /
&OBST XB=-0.83044,0.83044,-0.55711,0.55711,-0.0625,0 /
&OBST XB=-0.54211,0.54211,-0.84031,0.84031,-0.0625,0 /
&OBST XB=-0.84031,0.84031,-0.54211,0.54211,-0.0625,0 /
&OBST XB=-0.52711,0.52711,-0.8498,0.8498,-0.0625,0 /
&OBST XB=-0.8498,0.8498,-0.52711,0.52711,-0.0625,0 /
&OBST XB=-0.51211,0.51211,-0.85892,0.85892,-0.0625,0 /
&OBST XB=-0.85892,0.85892,-0.51211,0.51211,-0.0625,0 /
&OBST XB=-0.49711,0.49711,-0.86769,0.86769,-0.0625,0 /
&OBST XB=-0.86769,0.86769,-0.49711,0.49711,-0.0625,0 /
&OBST XB=-0.48211,0.48211,-0.87611,0.87611,-0.0625,0 /
&OBST XB=-0.87611,0.87611,-0.48211,0.48211,-0.0625,0 /
&OBST XB=-0.46711,0.46711,-0.8842,0.8842,-0.0625,0 /
```

```
&OBST XB=-0.8842,0.8842,-0.46711,0.46711,-0.0625,0 /
&OBST XB=-0.45211,0.45211,-0.89196,0.89196,-0.0625,0 /
&OBST XB=-0.89196,0.89196,-0.45211,0.45211,-0.0625,0 /
&OBST XB=-0.43711,0.43711,-0.89941,0.89941,-0.0625,0 /
&OBST XB=-0.89941,0.89941,-0.43711,0.43711,-0.0625,0 /
&OBST XB=-0.42211,0.42211,-0.90655,0.90655,-0.0625,0 /
&OBST XB=-0.90655,0.90655,-0.42211,0.42211,-0.0625,0 /
&OBST XB=-0.40711,0.40711,-0.91338,0.91338,-0.0625,0 /
&OBST XB=-0.91338,0.91338,-0.40711,0.40711,-0.0625,0 /
&OBST XB=-0.39211,0.39211,-0.91992,0.91992,-0.0625,0 /
&OBST XB=-0.91992,0.91992,-0.39211,0.39211,-0.0625,0 /
&OBST XB=-0.37711,0.37711,-0.92617,0.92617,-0.0625,0 /
&OBST XB=-0.92617,0.92617,-0.37711,0.37711,-0.0625,0 /
&OBST XB=-0.36211,0.36211,-0.93214,0.93214,-0.0625,0 /
&OBST XB=-0.93214,0.93214,-0.36211,0.36211,-0.0625,0 /
&OBST XB=-0.34711,0.34711,-0.93783,0.93783,-0.0625,0 /
&OBST XB=-0.93783,0.93783,-0.34711,0.34711,-0.0625,0 /
&OBST XB=-0.33211,0.33211,-0.94324,0.94324,-0.0625,0 /
&OBST XB=-0.94324,0.94324,-0.33211,0.33211,-0.0625,0 /
&OBST XB=-0.31711,0.31711,-0.94839,0.94839,-0.0625,0 /
&OBST XB=-0.94839,0.94839,-0.31711,0.31711,-0.0625,0 /
&OBST XB=-0.30211,0.30211,-0.95327,0.95327,-0.0625,0 /
&OBST XB=-0.95327,0.95327,-0.30211,0.30211,-0.0625,0 /
&OBST XB=-0.28711,0.28711,-0.9579,0.9579,-0.0625,0 /
&OBST XB=-0.9579,0.9579,-0.28711,0.28711,-0.0625,0 /
&OBST XB=-0.27211,0.27211,-0.96227,0.96227,-0.0625,0 /
&OBST XB=-0.96227,0.96227,-0.27211,0.27211,-0.0625,0 /
&OBST XB=-0.25711,0.25711,-0.96638,0.96638,-0.0625,0 /
&OBST XB=-0.96638,0.96638,-0.25711,0.25711,-0.0625,0 /
&OBST XB=-0.24211,0.24211,-0.97025,0.97025,-0.0625,0 /
&OBST XB=-0.97025,0.97025,-0.24211,0.24211,-0.0625,0 /
&OBST XB=-0.22711,0.22711,-0.97387,0.97387,-0.0625,0 /
&OBST XB=-0.97387,0.97387,-0.22711,0.22711,-0.0625,0 /
&OBST XB=-0.21211,0.21211,-0.97725,0.97725,-0.0625,0 /
&OBST XB=-0.97725,0.97725,-0.21211,0.21211,-0.0625,0 /
&OBST XB=-0.19711,0.19711,-0.98038,0.98038,-0.0625,0 /
&OBST XB=-0.98038,0.98038,-0.19711,0.19711,-0.0625,0 /
&OBST XB=-0.18211,0.18211,-0.98328,0.98328,-0.0625,0 /
&OBST XB=-0.98328,0.98328,-0.18211,0.18211,-0.0625,0 /
&OBST XB=-0.16711,0.16711,-0.98594,0.98594,-0.0625,0 /
&OBST XB=-0.98594,0.98594,-0.16711,0.16711,-0.0625,0 /
&OBST XB=-0.15211,0.15211,-0.98836,0.98836,-0.0625,0 /
&OBST XB=-0.98836,0.98836,-0.15211,0.15211,-0.0625,0 /
```

```
&OBST XB=−0.13711,0.13711,−0.99056,0.99056,−0.0625,0  /
&OBST XB=−0.99056,0.99056,−0.13711,0.13711,−0.0625,0  /
&OBST XB=−0.12211,0.12211,−0.99252,0.99252,−0.0625,0  /
&OBST XB=−0.99252,0.99252,−0.12211,0.12211,−0.0625,0  /
&OBST XB=−0.10711,0.10711,−0.99425,0.99425,−0.0625,0  /
&OBST XB=−0.99425,0.99425,−0.10711,0.10711,−0.0625,0  /
&OBST XB=−0.092107,0.092107,−0.99575,0.99575,−0.0625,0  /
&OBST XB=−0.99575,0.99575,−0.092107,0.092107,−0.0625,0  /
&OBST XB=−0.077107,0.077107,−0.99702,0.99702,−0.0625,0  /
&OBST XB=−0.99702,0.99702,−0.077107,0.077107,−0.0625,0  /
&OBST XB=−0.062107,0.062107,−0.99807,0.99807,−0.0625,0  /
&OBST XB=−0.99807,0.99807,−0.062107,0.062107,−0.0625,0  /
&OBST XB=−0.047107,0.047107,−0.99889,0.99889,−0.0625,0  /
&OBST XB=−0.99889,0.99889,−0.047107,0.047107,−0.0625,0  /
&OBST XB=−0.032107,0.032107,−0.99948,0.99948,−0.0625,0  /
&OBST XB=−0.99948,0.99948,−0.032107,0.032107,−0.0625,0  /
&OBST XB=−0.017107,0.017107,−0.99985,0.99985,−0.0625,0  /
&OBST XB=−0.99985,0.99985,−0.017107,0.017107,−0.0625,0  /
&OBST XB=−0.0021068,0.0021068,−1,1,−0.0625,0  /
&OBST XB=−1,1,−0.0021068,0.0021068,−0.0625,0  /

&HOLE XB=−0.35355,0.35355,−0.35355,0.35355,−.125,.1/
&HOLE XB=−0.35355,0.35355,−0.35355,0.35355,−.125,.1/
&HOLE XB=−0.33755,0.33755,−0.36886,0.36886,−.125,.1/
&HOLE XB=−0.36886,0.36886,−0.33755,0.33755,−.125,.1/
&HOLE XB=−0.32155,0.32155,−0.38289,0.38289,−.125,.1/
&HOLE XB=−0.38289,0.38289,−0.32155,0.32155,−.125,.1/
&HOLE XB=−0.30555,0.30555,−0.39577,0.39577,−.125,.1/
&HOLE XB=−0.39577,0.39577,−0.30555,0.30555,−.125,.1/
&HOLE XB=−0.28955,0.28955,−0.40763,0.40763,−.125,.1/
&HOLE XB=−0.40763,0.40763,−0.28955,0.28955,−.125,.1/
&HOLE XB=−0.27355,0.27355,−0.41853,0.41853,−.125,.1/
&HOLE XB=−0.41853,0.41853,−0.27355,0.27355,−.125,.1/
&HOLE XB=−0.25755,0.25755,−0.42856,0.42856,−.125,.1/
&HOLE XB=−0.42856,0.42856,−0.25755,0.25755,−.125,.1/
&HOLE XB=−0.24155,0.24155,−0.43778,0.43778,−.125,.1/
&HOLE XB=−0.43778,0.43778,−0.24155,0.24155,−.125,.1/
&HOLE XB=−0.22555,0.22555,−0.44623,0.44623,−.125,.1/
&HOLE XB=−0.44623,0.44623,−0.22555,0.22555,−.125,.1/
&HOLE XB=−0.20955,0.20955,−0.45397,0.45397,−.125,.1/
&HOLE XB=−0.45397,0.45397,−0.20955,0.20955,−.125,.1/
&HOLE XB=−0.19355,0.19355,−0.46102,0.46102,−.125,.1/
&HOLE XB=−0.46102,0.46102,−0.19355,0.19355,−.125,.1/
```

```
&HOLE XB=-0.17755,0.17755,-0.46741,0.46741,-.125,.1/
&HOLE XB=-0.46741,0.46741,-0.17755,0.17755,-.125,.1/
&HOLE XB=-0.16155,0.16155,-0.47318,0.47318,-.125,.1/
&HOLE XB=-0.47318,0.47318,-0.16155,0.16155,-.125,.1/
&HOLE XB=-0.14555,0.14555,-0.47835,0.47835,-.125,.1/
&HOLE XB=-0.47835,0.47835,-0.14555,0.14555,-.125,.1/
&HOLE XB=-0.12955,0.12955,-0.48292,0.48292,-.125,.1/
&HOLE XB=-0.48292,0.48292,-0.12955,0.12955,-.125,.1/
&HOLE XB=-0.11355,0.11355,-0.48693,0.48693,-.125,.1/
&HOLE XB=-0.48693,0.48693,-0.11355,0.11355,-.125,.1/
&HOLE XB=-0.097553,0.097553,-0.49039,0.49039,-.125,.1/
&HOLE XB=-0.49039,0.49039,-0.097553,0.097553,-.125,.1/
&HOLE XB=-0.081553,0.081553,-0.4933,0.4933,-.125,.1/
&HOLE XB=-0.4933,0.4933,-0.081553,0.081553,-.125,.1/
&HOLE XB=-0.065553,0.065553,-0.49568,0.49568,-.125,.1/
&HOLE XB=-0.49568,0.49568,-0.065553,0.065553,-.125,.1/
&HOLE XB=-0.049553,0.049553,-0.49754,0.49754,-.125,.1/
&HOLE XB=-0.49754,0.49754,-0.049553,0.049553,-.125,.1/
&HOLE XB=-0.033553,0.033553,-0.49887,0.49887,-.125,.1/
&HOLE XB=-0.49887,0.49887,-0.033553,0.033553,-.125,.1/
&HOLE XB=-0.017553,0.017553,-0.49969,0.49969,-.125,.1/
&HOLE XB=-0.49969,0.49969,-0.017553,0.017553,-.125,.1/
&HOLE XB=-0.0015534,0.0015534,-0.5,0.5,-.125,.1/
&HOLE XB=-0.5,0.5,-0.0015534,0.0015534,-.125,.1/

Radial profiles of velocity and mass fraction

&DEVC XB=-0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='W-VELOCITY', ID
    ='Wp3', POINTS=21, X_ID='x' /
&DEVC XB=-0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='W-VELOCITY', ID
    ='Wp5', POINTS=21, HIDE_COORDINATES=.TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='W-VELOCITY', ID
    ='Wp9', POINTS=21, HIDE_COORDINATES=.TRUE. /

&DEVC XB=-0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='U-VELOCITY', ID
    ='Up3', POINTS=21, HIDE_COORDINATES=.TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='U-VELOCITY', ID
    ='Up5', POINTS=21, HIDE_COORDINATES=.TRUE. /
&DEVC XB=-0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='U-VELOCITY', ID
    ='Up9', POINTS=21, HIDE_COORDINATES=.TRUE. /

&DEVC XB=-0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='W-VELOCITY', ID
    ='Wp3_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='W-VELOCITY', ID
    ='Wp5_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='W-VELOCITY', ID
    ='Wp9_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /

&DEVC XB= -0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='U-VELOCITY', ID
    ='Up3_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='U-VELOCITY', ID
    ='Up5_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='U-VELOCITY', ID
    ='Up9_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /

&TAIL /
```

A.3 Sandia test 24

```
&HEAD CHID='Sandia_CH4_1m_Test24_dx1p5cm', TITLE='Sandia 1m med
     flow rate methane pool fire (Test 24), 1.5 cm resolution' /

&MULT ID='mesh array', DX=1.5,DY=1.5,DZ=1.0, I_UPPER=1,J_UPPER
    =1,K_UPPER=3 /
&MESH IJK=96,96,64, XB= -1.5,0.0,-1.5,0.0,-.0625,0.9375, MULT_ID
    ='mesh array' /

&TIME T_END=20. /
&MISC TMPA=17.
        P_INF=81300.
        TURBULENCE_MODEL='VREMAN' /

&REAC FUEL='METHANE'
        HEAT_OF_COMBUSTION=49728.
        CRITICAL_FLAME_TEMPERATURE=1207 /

&RADI RADIATIVE_FRACTION=0.2 /
&DUMP SIG_FIGS=4, SIG_FIGS_EXP=2 /
&MATL ID='STEEL', CONDUCTIVITY=54., SPECIFIC_HEAT=0.465,
    DENSITY=7850., EMISSIVITY=0.9 /
```

```
&SURF ID='PLATE', COLOR='GRAY', DEFAULT=.TRUE., MATL_ID='STEEL
    ', THICKNESS=0.025 /

&SURF ID='POOL',MASS_FLUX(1)=0.053,SPEC_ID(1)='METHANE',
   TMP_FRONT=15. /

&VENT XB=-0.5,0.5,-0.5,0.5,-.0625,-.0625,COLOR='BLUE',SURF_ID='
   POOL'/

&VENT XB=-1.5,-.5,-1.5,1.5,-.0625,-.0625,SURF_ID='OPEN'/
&VENT XB=.5,1.5,-1.5,1.5,-.0625,-.0625,SURF_ID='OPEN'/
&VENT XB=-1.5,1.5,-1.5,-.5,-.0625,-.0625,SURF_ID='OPEN'/
&VENT XB=-1.5,1.5,.5,1.5,-.0625,-.0625,SURF_ID='OPEN'/

&VENT MB='ZMAX',SURF_ID='OPEN'/
&VENT MB='YMIN',SURF_ID='OPEN'/
&VENT MB='YMAX',SURF_ID='OPEN'/
&VENT MB='XMIN',SURF_ID='OPEN'/
&VENT MB='XMAX',SURF_ID='OPEN'/

&SLCF PBY=-.0625, QUANTITY='VELOCITY', VECTOR=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='DENSITY', CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='TEMPERATURE', CELL_CENTERED=.TRUE.
    /
&SLCF PBY=-.0625, QUANTITY='HRRPUV', CELL_CENTERED=.TRUE. /
/&SLCF PBY=-.0625, QUANTITY='HRRPUA', CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='MASS FRACTION', SPEC_ID='METHANE',
   CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='C_SMAG', CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='TURBULENCE RESOLUTION',
   CELL_CENTERED=.TRUE. /
&SLCF PBY=-.0625, QUANTITY='WAVELET ERROR', QUANTITY2='MASS
   FRACTION', SPEC_ID='METHANE', CELL_CENTERED=.TRUE. /

&OBST XB=-0.70711,0.70711,-0.70711,0.70711,-0.0625,0 /
&OBST XB=-0.70711,0.70711,-0.70711,0.70711,-0.0625,0 /
&OBST XB=-0.69211,0.69211,-0.7218,0.7218,-0.0625,0 /
&OBST XB=-0.7218,0.7218,-0.69211,0.69211,-0.0625,0 /
&OBST XB=-0.67711,0.67711,-0.73588,0.73588,-0.0625,0 /
&OBST XB=-0.73588,0.73588,-0.67711,0.67711,-0.0625,0 /
&OBST XB=-0.66211,0.66211,-0.74941,0.74941,-0.0625,0 /
&OBST XB=-0.74941,0.74941,-0.66211,0.66211,-0.0625,0 /
&OBST XB=-0.64711,0.64711,-0.7624,0.7624,-0.0625,0 /
```

```
&OBST XB=−0.7624,0.7624,−0.64711,0.64711,−0.0625,0 /
&OBST XB=−0.63211,0.63211,−0.77488,0.77488,−0.0625,0 /
&OBST XB=−0.77488,0.77488,−0.63211,0.63211,−0.0625,0 /
&OBST XB=−0.61711,0.61711,−0.78688,0.78688,−0.0625,0 /
&OBST XB=−0.78688,0.78688,−0.61711,0.61711,−0.0625,0 /
&OBST XB=−0.60211,0.60211,−0.79842,0.79842,−0.0625,0 /
&OBST XB=−0.79842,0.79842,−0.60211,0.60211,−0.0625,0 /
&OBST XB=−0.58711,0.58711,−0.80951,0.80951,−0.0625,0 /
&OBST XB=−0.80951,0.80951,−0.58711,0.58711,−0.0625,0 /
&OBST XB=−0.57211,0.57211,−0.82018,0.82018,−0.0625,0 /
&OBST XB=−0.82018,0.82018,−0.57211,0.57211,−0.0625,0 /
&OBST XB=−0.55711,0.55711,−0.83044,0.83044,−0.0625,0 /
&OBST XB=−0.83044,0.83044,−0.55711,0.55711,−0.0625,0 /
&OBST XB=−0.54211,0.54211,−0.84031,0.84031,−0.0625,0 /
&OBST XB=−0.84031,0.84031,−0.54211,0.54211,−0.0625,0 /
&OBST XB=−0.52711,0.52711,−0.8498,0.8498,−0.0625,0 /
&OBST XB=−0.8498,0.8498,−0.52711,0.52711,−0.0625,0 /
&OBST XB=−0.51211,0.51211,−0.85892,0.85892,−0.0625,0 /
&OBST XB=−0.85892,0.85892,−0.51211,0.51211,−0.0625,0 /
&OBST XB=−0.49711,0.49711,−0.86769,0.86769,−0.0625,0 /
&OBST XB=−0.86769,0.86769,−0.49711,0.49711,−0.0625,0 /
&OBST XB=−0.48211,0.48211,−0.87611,0.87611,−0.0625,0 /
&OBST XB=−0.87611,0.87611,−0.48211,0.48211,−0.0625,0 /
&OBST XB=−0.46711,0.46711,−0.8842,0.8842,−0.0625,0 /
&OBST XB=−0.8842,0.8842,−0.46711,0.46711,−0.0625,0 /
&OBST XB=−0.45211,0.45211,−0.89196,0.89196,−0.0625,0 /
&OBST XB=−0.89196,0.89196,−0.45211,0.45211,−0.0625,0 /
&OBST XB=−0.43711,0.43711,−0.89941,0.89941,−0.0625,0 /
&OBST XB=−0.89941,0.89941,−0.43711,0.43711,−0.0625,0 /
&OBST XB=−0.42211,0.42211,−0.90655,0.90655,−0.0625,0 /
&OBST XB=−0.90655,0.90655,−0.42211,0.42211,−0.0625,0 /
&OBST XB=−0.40711,0.40711,−0.91338,0.91338,−0.0625,0 /
&OBST XB=−0.91338,0.91338,−0.40711,0.40711,−0.0625,0 /
&OBST XB=−0.39211,0.39211,−0.91992,0.91992,−0.0625,0 /
&OBST XB=−0.91992,0.91992,−0.39211,0.39211,−0.0625,0 /
&OBST XB=−0.37711,0.37711,−0.92617,0.92617,−0.0625,0 /
&OBST XB=−0.92617,0.92617,−0.37711,0.37711,−0.0625,0 /
&OBST XB=−0.36211,0.36211,−0.93214,0.93214,−0.0625,0 /
&OBST XB=−0.93214,0.93214,−0.36211,0.36211,−0.0625,0 /
&OBST XB=−0.34711,0.34711,−0.93783,0.93783,−0.0625,0 /
&OBST XB=−0.93783,0.93783,−0.34711,0.34711,−0.0625,0 /
&OBST XB=−0.33211,0.33211,−0.94324,0.94324,−0.0625,0 /
&OBST XB=−0.94324,0.94324,−0.33211,0.33211,−0.0625,0 /
```

```
&OBST XB=−0.31711,0.31711,−0.94839,0.94839,−0.0625,0 /
&OBST XB=−0.94839,0.94839,−0.31711,0.31711,−0.0625,0 /
&OBST XB=−0.30211,0.30211,−0.95327,0.95327,−0.0625,0 /
&OBST XB=−0.95327,0.95327,−0.30211,0.30211,−0.0625,0 /
&OBST XB=−0.28711,0.28711,−0.9579,0.9579,−0.0625,0 /
&OBST XB=−0.9579,0.9579,−0.28711,0.28711,−0.0625,0 /
&OBST XB=−0.27211,0.27211,−0.96227,0.96227,−0.0625,0 /
&OBST XB=−0.96227,0.96227,−0.27211,0.27211,−0.0625,0 /
&OBST XB=−0.25711,0.25711,−0.96638,0.96638,−0.0625,0 /
&OBST XB=−0.96638,0.96638,−0.25711,0.25711,−0.0625,0 /
&OBST XB=−0.24211,0.24211,−0.97025,0.97025,−0.0625,0 /
&OBST XB=−0.97025,0.97025,−0.24211,0.24211,−0.0625,0 /
&OBST XB=−0.22711,0.22711,−0.97387,0.97387,−0.0625,0 /
&OBST XB=−0.97387,0.97387,−0.22711,0.22711,−0.0625,0 /
&OBST XB=−0.21211,0.21211,−0.97725,0.97725,−0.0625,0 /
&OBST XB=−0.97725,0.97725,−0.21211,0.21211,−0.0625,0 /
&OBST XB=−0.19711,0.19711,−0.98038,0.98038,−0.0625,0 /
&OBST XB=−0.98038,0.98038,−0.19711,0.19711,−0.0625,0 /
&OBST XB=−0.18211,0.18211,−0.98328,0.98328,−0.0625,0 /
&OBST XB=−0.98328,0.98328,−0.18211,0.18211,−0.0625,0 /
&OBST XB=−0.16711,0.16711,−0.98594,0.98594,−0.0625,0 /
&OBST XB=−0.98594,0.98594,−0.16711,0.16711,−0.0625,0 /
&OBST XB=−0.15211,0.15211,−0.98836,0.98836,−0.0625,0 /
&OBST XB=−0.98836,0.98836,−0.15211,0.15211,−0.0625,0 /
&OBST XB=−0.13711,0.13711,−0.99056,0.99056,−0.0625,0 /
&OBST XB=−0.99056,0.99056,−0.13711,0.13711,−0.0625,0 /
&OBST XB=−0.12211,0.12211,−0.99252,0.99252,−0.0625,0 /
&OBST XB=−0.99252,0.99252,−0.12211,0.12211,−0.0625,0 /
&OBST XB=−0.10711,0.10711,−0.99425,0.99425,−0.0625,0 /
&OBST XB=−0.99425,0.99425,−0.10711,0.10711,−0.0625,0 /
&OBST XB=−0.092107,0.092107,−0.99575,0.99575,−0.0625,0 /
&OBST XB=−0.99575,0.99575,−0.092107,0.092107,−0.0625,0 /
&OBST XB=−0.077107,0.077107,−0.99702,0.99702,−0.0625,0 /
&OBST XB=−0.99702,0.99702,−0.077107,0.077107,−0.0625,0 /
&OBST XB=−0.062107,0.062107,−0.99807,0.99807,−0.0625,0 /
&OBST XB=−0.99807,0.99807,−0.062107,0.062107,−0.0625,0 /
&OBST XB=−0.047107,0.047107,−0.99889,0.99889,−0.0625,0 /
&OBST XB=−0.99889,0.99889,−0.047107,0.047107,−0.0625,0 /
&OBST XB=−0.032107,0.032107,−0.99948,0.99948,−0.0625,0 /
&OBST XB=−0.99948,0.99948,−0.032107,0.032107,−0.0625,0 /
&OBST XB=−0.017107,0.017107,−0.99985,0.99985,−0.0625,0 /
&OBST XB=−0.99985,0.99985,−0.017107,0.017107,−0.0625,0 /
&OBST XB=−0.0021068,0.0021068,−1,1,−0.0625,0 /
```

```
&OBST XB=-1,1,-0.0021068,0.0021068,-0.0625,0  /

&HOLE XB=-0.35355,0.35355,-0.35355,0.35355,-.125,.1/
&HOLE XB=-0.35355,0.35355,-0.35355,0.35355,-.125,.1/
&HOLE XB=-0.33755,0.33755,-0.36886,0.36886,-.125,.1/
&HOLE XB=-0.36886,0.36886,-0.33755,0.33755,-.125,.1/
&HOLE XB=-0.32155,0.32155,-0.38289,0.38289,-.125,.1/
&HOLE XB=-0.38289,0.38289,-0.32155,0.32155,-.125,.1/
&HOLE XB=-0.30555,0.30555,-0.39577,0.39577,-.125,.1/
&HOLE XB=-0.39577,0.39577,-0.30555,0.30555,-.125,.1/
&HOLE XB=-0.28955,0.28955,-0.40763,0.40763,-.125,.1/
&HOLE XB=-0.40763,0.40763,-0.28955,0.28955,-.125,.1/
&HOLE XB=-0.27355,0.27355,-0.41853,0.41853,-.125,.1/
&HOLE XB=-0.41853,0.41853,-0.27355,0.27355,-.125,.1/
&HOLE XB=-0.25755,0.25755,-0.42856,0.42856,-.125,.1/
&HOLE XB=-0.42856,0.42856,-0.25755,0.25755,-.125,.1/
&HOLE XB=-0.24155,0.24155,-0.43778,0.43778,-.125,.1/
&HOLE XB=-0.43778,0.43778,-0.24155,0.24155,-.125,.1/
&HOLE XB=-0.22555,0.22555,-0.44623,0.44623,-.125,.1/
&HOLE XB=-0.44623,0.44623,-0.22555,0.22555,-.125,.1/
&HOLE XB=-0.20955,0.20955,-0.45397,0.45397,-.125,.1/
&HOLE XB=-0.45397,0.45397,-0.20955,0.20955,-.125,.1/
&HOLE XB=-0.19355,0.19355,-0.46102,0.46102,-.125,.1/
&HOLE XB=-0.46102,0.46102,-0.19355,0.19355,-.125,.1/
&HOLE XB=-0.17755,0.17755,-0.46741,0.46741,-.125,.1/
&HOLE XB=-0.46741,0.46741,-0.17755,0.17755,-.125,.1/
&HOLE XB=-0.16155,0.16155,-0.47318,0.47318,-.125,.1/
&HOLE XB=-0.47318,0.47318,-0.16155,0.16155,-.125,.1/
&HOLE XB=-0.14555,0.14555,-0.47835,0.47835,-.125,.1/
&HOLE XB=-0.47835,0.47835,-0.14555,0.14555,-.125,.1/
&HOLE XB=-0.12955,0.12955,-0.48292,0.48292,-.125,.1/
&HOLE XB=-0.48292,0.48292,-0.12955,0.12955,-.125,.1/
&HOLE XB=-0.11355,0.11355,-0.48693,0.48693,-.125,.1/
&HOLE XB=-0.48693,0.48693,-0.11355,0.11355,-.125,.1/
&HOLE XB=-0.097553,0.097553,-0.49039,0.49039,-.125,.1/
&HOLE XB=-0.49039,0.49039,-0.097553,0.097553,-.125,.1/
&HOLE XB=-0.081553,0.081553,-0.4933,0.4933,-.125,.1/
&HOLE XB=-0.4933,0.4933,-0.081553,0.081553,-.125,.1/
&HOLE XB=-0.065553,0.065553,-0.49568,0.49568,-.125,.1/
&HOLE XB=-0.49568,0.49568,-0.065553,0.065553,-.125,.1/
&HOLE XB=-0.049553,0.049553,-0.49754,0.49754,-.125,.1/
&HOLE XB=-0.49754,0.49754,-0.049553,0.049553,-.125,.1/
&HOLE XB=-0.033553,0.033553,-0.49887,0.49887,-.125,.1/
```

```
&HOLE XB= -0.49887,0.49887,-0.033553,0.033553,-.125,.1/
&HOLE XB= -0.017553,0.017553,-0.49969,0.49969,-.125,.1/
&HOLE XB= -0.49969,0.49969,-0.017553,0.017553,-.125,.1/
&HOLE XB= -0.0015534,0.0015534,-0.5,0.5,-.125,.1/
&HOLE XB= -0.5,0.5,-0.0015534,0.0015534,-.125,.1/

Radial profiles of velocity and mass fraction

&DEVC XB= -0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='W-VELOCITY', ID
    ='Wp3', POINTS=21, X_ID='x' /
&DEVC XB= -0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='W-VELOCITY', ID
    ='Wp5', POINTS=21, HIDE_COORDINATES=.TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='W-VELOCITY', ID
    ='Wp9', POINTS=21, HIDE_COORDINATES=.TRUE. /

&DEVC XB= -0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='U-VELOCITY', ID
    ='Up3', POINTS=21, HIDE_COORDINATES=.TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='U-VELOCITY', ID
    ='Up5', POINTS=21, HIDE_COORDINATES=.TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='U-VELOCITY', ID
    ='Up9', POINTS=21, HIDE_COORDINATES=.TRUE. /

&DEVC XB= -0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='W-VELOCITY', ID
    ='Wp3_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='W-VELOCITY', ID
    ='Wp5_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='W-VELOCITY', ID
    ='Wp9_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /

&DEVC XB= -0.50,0.50,0.0,0.0,0.3,0.3, QUANTITY='U-VELOCITY', ID
    ='Up3_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.5,0.5, QUANTITY='U-VELOCITY', ID
    ='Up5_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /
&DEVC XB= -0.50,0.50,0.0,0.0,0.9,0.9, QUANTITY='U-VELOCITY', ID
    ='Up9_rms', STATISTICS='RMS', POINTS=21, HIDE_COORDINATES=.
    TRUE. /

&TAIL /
```

A.4 Heskestad

```
&HEAD CHID='Qs=10000_RI=20', TITLE='Flame Height Test, Q
    *=10000' /
&MESH IJK=65,65,160, XB=-72.0,72.0,-72.0,72.0,-18.0,342. /
&MISC TURBULENCE_MODEL='VREMAN'/
&TIME T_END=200. /

&REAC FUEL='PROPANE', C=3., H=8., SOOT_YIELD=0.015 /
&SURF ID='burner', HRRPUA=15127250., COLOR='RED' /
&OBST XB=-0.5,0.5,-0.5,0.5,-5.0,0.0, SURF_IDS='burner','INERT
    ','INERT', THICKEN=.TRUE. /

&VENT MB='XMIN', SURF_ID='OPEN' /
&VENT MB='XMAX', SURF_ID='OPEN' /
&VENT MB='YMIN', SURF_ID='OPEN' /
&VENT MB='YMAX', SURF_ID='OPEN' /
&VENT MB='ZMIN', SURF_ID='OPEN' /
&VENT MB='ZMAX', SURF_ID='OPEN' /

&SLCF PBY=0.,QUANTITY='TEMPERATURE',VECTOR=.TRUE. /
&SLCF PBY=0.,QUANTITY='HRRPUV' /
&SLCF PBY=0.,QUANTITY='MIXING TIME' /
&DEVC XB=0.0,0.0,0.0,0.0,1.12,340.77, QUANTITY='HRRPUL', POINTS
    =152, Z_ID='Height', ID='HRRPUL' /
&TAIL /
```

A.5 McCaffery

```
&HEAD CHID='McCaffrey_57_kW_fine', TITLE='McCaffrey, NBSIR
    79-1910, 57 kW Natural Gas' /
&MISC TURBULENCE_MODEL='VREMAN'/
&MULT ID='mesh', DX=0.62, DY=0.62, DZ=1.55, I_UPPER=2, J_UPPER
    =2, K_UPPER=3 /
&MESH IJK=40,40,100, XB=-0.93,-0.31,-0.93,-0.31,-0.248,1.302,
    MULT_ID='mesh' /

&TIME T_END=30. /

&SURF ID='burner', HRRPUA=639., TMP_FRONT=100., COLOR='RED' /
&OBST XB=-.15,0.15,-.15,0.15,-.10,0.00,SURF_IDS='burner','INERT
    ','INERT' /
```

```
&REAC FUEL='METHANE'
        C=1.
        H=4.
        CO_YIELD=0.0
        SOOT_YIELD=0.0  /

&RADI RADIATIVE_FRACTION=0.20  /

&VENT MB='XMIN', SURF_ID='OPEN'  /
&VENT MB='XMAX', SURF_ID='OPEN'  /
&VENT MB='YMIN', SURF_ID='OPEN'  /
&VENT MB='YMAX', SURF_ID='OPEN'  /
&VENT MB='ZMAX', SURF_ID='OPEN'  /
&VENT MB='ZMIN', SURF_ID='OPEN'  /

&DEVC ID='temp20', XB=0.00,0.00,0.00,0.00,0.0155,5.9365, POINTS
    =192, QUANTITY='TEMPERATURE', Z_ID='Height'  /
&DEVC ID='velo20', XB=0.00,0.00,0.00,0.00,0.0155,5.9365, POINTS
    =192, QUANTITY='W-VELOCITY', HIDE_COORDINATES=.TRUE.  /

&DEVC XYZ=0,0,0.3, QUANTITY='W-VELOCITY'/

&SLCF PBY=0.0, QUANTITY='WAVELET ERROR', QUANTITY2='MASS
    FRACTION', SPEC_ID='METHANE'/
&SLCF PBY=0.0, QUANTITY='WAVELET ERROR', QUANTITY2='HRRPUV'/
&SLCF PBY=0.0, QUANTITY='WAVELET ERROR', QUANTITY2='TEMPERATURE
    '/
&SLCF PBY=0.0, QUANTITY='TURBULENCE RESOLUTION'  /

&SLCF PBY=0.0,QUANTITY='TEMPERATURE',VECTOR=.TRUE.  /
&SLCF PBY=0.0,QUANTITY='HRRPUV'    /

&TAIL /
```

Appendix B

Implemented code

B.1 fire.f90

```
MODULE FIRE

! Compute combustion

USE PRECISION_PARAMETERS
USE GLOBAL_CONSTANTS
USE MESH_POINTERS
USE COMP_FUNCTIONS, ONLY: SECOND

IMPLICIT NONE
PRIVATE
CHARACTER(255), PARAMETER :: fireid='$Id: fire.f90 10216
    2012-03-08 16:22:22Z craigweinschenk $'
CHARACTER(255), PARAMETER :: firerev='$Revision: 10216 $'
CHARACTER(255), PARAMETER :: firedate='$Date: 2012-03-08
    17:22:22 +0100 (to, 08 mar 2012) $'

TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
REAL(EB) :: Q_UPPER

PUBLIC COMBUSTION, GET_REV_fire

CONTAINS


SUBROUTINE COMBUSTION(NM)

INTEGER, INTENT(IN) :: NM
```

```
REAL(EB) :: TNOW

IF (EVACUATION_ONLY(NM)) RETURN

TNOW=SECOND()

IF (INIT_HRRPUV) RETURN

CALL POINT_TO_MESH(NM)

!CALL COMPUTE_STRAIN_RATE(NM)

! Upper bounds on local HRR per unit volume

Q_UPPER = HRRPUA_SHEET/CELL_SIZE + HRRPUV_AVERAGE

! Call combustion ODE solver
CALL COMBUSTION_GENERAL

TUSED(10,NM)=TUSED(10,NM)+SECOND()-TNOW

END SUBROUTINE COMBUSTION


SUBROUTINE COMBUSTION_GENERAL

! Generic combustion routine for multi step reactions with
    kinetics either mixing controlled, finite rate,
! or a temperature threshhold mixed approach

USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT,
    GET_MASS_FRACTION_ALL,GET_SPECIFIC_HEAT,GET_MOLECULAR_WEIGHT
    , &
                                        GET_SENSIBLE_ENTHALPY_DIFF,
                                           GET_MASS_FRACTION !ADDED
INTEGER :: I,J,K,NS,NR,II,JJ,KK,IIG,JJG,KKG,IW,N
REAL(EB) :: ZZ_GET(0:N_TRACKED_SPECIES),ZZ_MIN=1.E-10_EB,DZZ(0:
    N_TRACKED_SPECIES),CP,HDIFF,Y_O2,Y_FUEL,Y_PRODUCT !ADDED
LOGICAL :: DO_REACTION,REACTANTS_PRESENT,Q_EXISTS
TYPE (REACTION_TYPE),POINTER :: RN
TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM,SM0

Q          = 0._EB
```

```fortran
D_REACTION = 0._EB
Q_EXISTS = .FALSE.
SM0 => SPECIES_MIXTURE(0)

DO K=1,KBAR
   DO J=1,JBAR
      ILOOP: DO I=1,IBAR
         !Check to see if a reaction is possible
         IF (SOLID(CELL_INDEX(I,J,K))) CYCLE ILOOP
         ZZ_GET(1:N_TRACKED_SPECIES) = ZZ(I,J,K,1:
            N_TRACKED_SPECIES)
         ZZ_GET(0) = 1._EB - MIN(1._EB,SUM(ZZ_GET(1:
            N_TRACKED_SPECIES)))
         DO_REACTION = .FALSE.
         DO NR=1,N_REACTIONS
            RN=>REACTION(NR)
            REACTANTS_PRESENT = .TRUE.
            DO NS=0,N_TRACKED_SPECIES
               IF (RN%NU(NS)<0._EB .AND. ZZ_GET(NS) < ZZ_MIN)
                  THEN
                  REACTANTS_PRESENT = .FALSE.
                  EXIT
               ENDIF
            END DO
            IF (.NOT. DO_REACTION) DO_REACTION =
               REACTANTS_PRESENT
         END DO
         IF (.NOT. DO_REACTION) CYCLE ILOOP
         DZZ(1:N_TRACKED_SPECIES) = ZZ_GET(1:N_TRACKED_SPECIES)
               ! store old ZZ for divergence term
         ! Easily allow for user selected ODE solver
         SELECT CASE (COMBUSTION_ODE)
            CASE(SINGLE_EXACT)
               !CALL ODE_EXACT(I,J,K,ZZ_GET,Q(I,J,K))
               CALL ODE_EXPLICIT_EULER(I,J,K,ZZ_GET,Q(I,J,K))
            CASE(EXPLICIT_EULER)
               CALL ODE_EXPLICIT_EULER(I,J,K,ZZ_GET,Q(I,J,K))
            CASE(RUNGE_KUTTA_2)
               CALL ODE_RUNGE_KUTTA_2(I,J,K,ZZ_GET,Q(I,J,K))
            CASE(RK2_RICHARDSON)
               CALL ODE_RK2_RICHARDSON(I,J,K,ZZ_GET,Q(I,J,K))
         END SELECT

```

```
            ! Update RSUM and ZZ
            IF (ABS(Q(I,J,K)) > ZERO_P) THEN
               Q_EXISTS = .TRUE.
               CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET,RSUM(I,J,K))
               TMP(I,J,K) = PBAR(K,PRESSURE_ZONE(I,J,K))/(RSUM(I,J
                  ,K)*RHO(I,J,K))
               ZZ(I,J,K,1:N_TRACKED_SPECIES) = ZZ_GET(1:
                  N_TRACKED_SPECIES)
               ! Divergence term
               DZZ(1:N_TRACKED_SPECIES) = ZZ_GET(1:
                  N_TRACKED_SPECIES) - DZZ(1:N_TRACKED_SPECIES)
               CALL GET_SPECIFIC_HEAT(ZZ_GET,CP,TMP(I,J,K))
               DO N=1,N_TRACKED_SPECIES
                  SM => SPECIES_MIXTURE(N)
                  CALL GET_SENSIBLE_ENTHALPY_DIFF(N,TMP(I,J,K),
                     HDIFF)
                  D_REACTION(I,J,K) = D_REACTION(I,J,K) + ( (SM%
                     RCON-SM0%RCON)/RSUM(I,J,K) - &
                                                    HDIFF
                                                      /(
                                                      CP*
                                                      TMP
                                                      (I,
                                                      J,K
                                                      ))
                                                      )*
                                                      DZZ
                                                      (N)
                                                      /DT
               ENDDO
            ENDIF

         ENDDO ILOOP
      ENDDO
   ENDDO

IF (.NOT. Q_EXISTS) RETURN

! Set Q in the ghost cell, just for better visualization.
DO IW=1,N_EXTERNAL_WALL_CELLS
   IF (WALL(IW)%BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY .AND. WALL
      (IW)%BOUNDARY_TYPE/=OPEN_BOUNDARY) CYCLE
   II  = WALL(IW)%ONE_D%II
```

```
    JJ   = WALL(IW)%ONE_D%JJ
    KK   = WALL(IW)%ONE_D%KK
    IIG  = WALL(IW)%ONE_D%IIG
    JJG  = WALL(IW)%ONE_D%JJG
    KKG  = WALL(IW)%ONE_D%KKG
    Q(II,JJ,KK) = Q(IIG,JJG,KKG)
ENDDO

END SUBROUTINE COMBUSTION_GENERAL

SUBROUTINE ODE_EXACT(I,J,K,ZZ_GET,Q_NEW)
INTEGER,INTENT(IN) :: I,J,K
REAL(EB),INTENT(OUT) :: Q_NEW
REAL(EB),INTENT(INOUT) :: ZZ_GET(0:N_TRACKED_SPECIES)
REAL(EB) :: DZF,Q_BOUND_1,Q_BOUND_2,RATE_CONSTANT,Z_LIMITER,
    REACTANT_MIN,DT2
LOGICAL :: MIN_FOUND
INTEGER :: NS
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

Q_NEW = 0._EB
RN=>REACTION(1)
CALL COMPUTE_RATE_CONSTANT(1,RN%MODE,1,0._EB,RATE_CONSTANT,
    ZZ_GET,I,J,K)

IF(RATE_CONSTANT < ZERO_P) RETURN

Z_LIMITER = RATE_CONSTANT*MIX_TIME(I,J,K)

DZF = -1._EB
!Check for reactant (i.e. fuel or oxidizer) limited combustion
MIN_FOUND = .FALSE.
REACTANT_MIN=1._EB
DO NS=0,N_TRACKED_SPECIES
    IF (RN%NU(NS) < -ZERO_P) &
        REACTANT_MIN = MIN(REACTANT_MIN,-ZZ_GET(NS)*
            SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW/(
            SPECIES_MIXTURE(NS)%MW*RN%NU(NS)))
    IF (ABS(Z_LIMITER - REACTANT_MIN) <= SPACING(Z_LIMITER))
        THEN
        MIN_FOUND = .TRUE.
        DZF = REACTANT_MIN*(1._EB-EXP(-DT/MIX_TIME(I,J,K)))
        EXIT
    ENDIF
ENDDO

!For product limited combsiton find time of switch from product limited to reactant limited (if it occurs)
!and do two step exact solution
IF (.NOT. MIN_FOUND) THEN
    DT2 = MIX_TIME(I,J,K)*LOG((Z_LIMITER+REACTANT_MIN)/(2._EB*Z_LIMITER))
    IF (DT2 < DT) THEN
        DZF = ZZ_GET(RN%FUEL_SMIX_INDEX) - Z_LIMITER*(EXP(DT2/MIX_TIME(I,J,K))-1._EB)
        REACTANT_MIN = REACTANT_MIN - DZF
        DZF = DZF + REACTANT_MIN*(1._EB-EXP(-(DT-DT2)/MIX_TIME(I,J,K)))
    ELSE
        DZF = ZZ_GET(RN%FUEL_SMIX_INDEX) - Z_LIMITER*(EXP(DT/MIX_TIME(I,J,K))-1._EB)
    ENDIF
ENDIF

DZF = MIN(DZF,ZZ_GET(RN%FUEL_SMIX_INDEX))

!****** TEMP OVERRIDE TO ENSURE SAME RESULTS AS PREVIOUS *******
!DZF = Z_LIMITER*(1._EB-EXP(-DT/MIX_TIME(I,J,K)))
!**************************************************************


Q_BOUND_1 = DZF*RHO(I,J,K)*RN%HEAT_OF_COMBUSTION/DT
Q_BOUND_2 = Q_UPPER
Q_NEW = MIN(Q_BOUND_1,Q_BOUND_2)
DZF = Q_NEW*DT/(RHO(I,J,K)*RN%HEAT_OF_COMBUSTION)

ZZ_GET = ZZ_GET + DZF*RN%NU*SPECIES_MIXTURE%MW/SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW

END SUBROUTINE ODE_EXACT


SUBROUTINE ODE_EXPLICIT_EULER(I,J,K,ZZ_GET,Q_OUT)
INTEGER,INTENT(IN) :: I,J,K
```

```
REAL(EB),INTENT(OUT) :: Q_OUT
REAL(EB),INTENT(INOUT) :: ZZ_GET(0:N_TRACKED_SPECIES)
REAL(EB) :: ZZ_0(0:N_TRACKED_SPECIES),ZZ_I(0:N_TRACKED_SPECIES)
      ,ZZ_N(0:N_TRACKED_SPECIES),DZZDT(0:N_TRACKED_SPECIES),&
                DT_ODE,DT_NEW,RATE_CONSTANT(1:N_REACTIONS),Q_NR(1:
                    N_REACTIONS),Q_SUM,DT_SUM
INTEGER :: NR,I_TS,NS
INTEGER, PARAMETER :: NODETS=20
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

Q_OUT = 0._EB
 ZZ_0 = MAX(0._EB,ZZ_GET)
 ZZ_I = ZZ_0
DT_ODE = DT/REAL(NODETS,EB)
DT_NEW = DT_ODE
DT_SUM = 0._EB
 I_TS = 1
ODE_LOOP: DO WHILE (DT_SUM < DT)
    DZZDT = 0._EB
    RATE_CONSTANT = 0._EB
    Q_NR = 0._EB
    REACTION_LOOP: DO NR = 1, N_REACTIONS
        RN => REACTION(NR)
        CALL COMPUTE_RATE_CONSTANT(NR,RN%MODE,I_TS,Q_OUT,
            RATE_CONSTANT(NR),ZZ_I,I,J,K)
        IF (RATE_CONSTANT(NR) < ZERO_P) CYCLE REACTION_LOOP
        Q_NR(NR) = RATE_CONSTANT(NR)*RN%HEAT_OF_COMBUSTION*RHO(I,
            J,K)
        DZZDT = DZZDT + RN%NU * SPECIES_MIXTURE%MW/
            SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW*RATE_CONSTANT(
            NR)
    END DO REACTION_LOOP
    IF (ALL(DZZDT < ZERO_P)) EXIT ODE_LOOP
    ZZ_N = ZZ_I + DZZDT * DT_NEW

    IF (ANY(ZZ_N < 0._EB)) THEN
        DO NS=0,N_TRACKED_SPECIES
              IF (ZZ_N(NS) < 0._EB .AND. ABS(DZZDT(NS))>ZERO_P)
                  DT_NEW = MIN(DT_NEW,-ZZ_I(NS)/DZZDT(NS))
        ENDDO
    ENDIF

    Q_SUM = SUM(Q_NR)
```

```
      IF (Q_OUT + Q_SUM*DT_NEW > Q_UPPER * DT) THEN
         DT_NEW = MAX(0._EB,(Q_UPPER * DT - Q_OUT))/Q_SUM
         Q_OUT = Q_OUT+Q_SUM*DT_NEW
         ZZ_I = ZZ_I + DZZDT * DT_NEW
         EXIT ODE_LOOP
      ENDIF
      Q_OUT = Q_OUT+Q_SUM*DT_NEW
      ZZ_I = ZZ_I + DZZDT * DT_NEW
      DT_SUM = DT_SUM + DT_NEW
      IF (DT_NEW < DT_ODE) DT_NEW = DT_ODE
      IF (DT_NEW + DT_SUM > DT) DT_NEW = DT - DT_SUM
      I_TS = I_TS + 1
ENDDO ODE_LOOP

ZZ_GET = ZZ_GET + ZZ_I - ZZ_0
Q_OUT = Q_OUT / DT

RETURN

END SUBROUTINE ODE_EXPLICIT_EULER


SUBROUTINE ODE_RUNGE_KUTTA_2(I,J,K,ZZ_GET,Q_OUT)
INTEGER,INTENT(IN) :: I,J,K
REAL(EB),INTENT(OUT) :: Q_OUT
REAL(EB),INTENT(INOUT) :: ZZ_GET(0:N_TRACKED_SPECIES)
REAL(EB) :: ZZ_0(0:N_TRACKED_SPECIES),ZZ_I(0:N_TRACKED_SPECIES),ZZ_N(0:N_TRACKED_SPECIES),&
                 DZZDT(0:N_TRACKED_SPECIES),DZZDT2(0:N_TRACKED_SPECIES),&
                 DT_ODE,DT_NEW,RATE_CONSTANT(1:N_REACTIONS),Q_NR(1:N_REACTIONS),Q_NR2(1:N_REACTIONS),Q_SUM,DT_SUM
INTEGER :: NR,I_TS,NS
INTEGER, PARAMETER :: NODETS=20
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()


Q_OUT = 0._EB
ZZ_0 = MAX(0._EB,ZZ_GET)
ZZ_I = ZZ_0
DT_ODE = DT/REAL(NODETS,EB)
DT_NEW = DT_ODE
DT_SUM = 0._EB
```

```
I_TS = 1
ODE_LOOP: DO WHILE (DT_SUM < DT)
    DZZDT = 0._EB
    DZZDT2 = 0._EB
    Q_NR = 0._EB
    Q_NR2 = 0._EB
    RATE_CONSTANT = 0._EB
    REACTION_LOOP: DO NR = 1, N_REACTIONS
        RN => REACTION(NR)
        CALL COMPUTE_RATE_CONSTANT(NR,RN%MODE,I_TS,Q_OUT,
            RATE_CONSTANT(NR),ZZ_I,I,J,K)
        IF (RATE_CONSTANT(NR) < ZERO_P) CYCLE REACTION_LOOP
        Q_NR(NR) = RATE_CONSTANT(NR)*RN%HEAT_OF_COMBUSTION*RHO(I,
            J,K)
        DZZDT = DZZDT + RN%NU * SPECIES_MIXTURE%MW/
            SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW*RATE_CONSTANT(
            NR)
    END DO REACTION_LOOP
    IF (ALL(DZZDT < ZERO_P)) EXIT ODE_LOOP
    ZZ_N = ZZ_I + DZZDT * DT_NEW

    IF (ANY(ZZ_N < 0._EB)) THEN
        DO NS=0,N_TRACKED_SPECIES
             IF (ZZ_N(NS) < 0._EB .AND. ABS(DZZDT(NS))>ZERO_P)
                 DT_NEW = MIN(DT_NEW,-ZZ_I(NS)/DZZDT(NS))
        ENDDO
    ENDIF

    ZZ_N = ZZ_I + DZZDT * DT_NEW

    REACTION_LOOP2: DO NR = 1, N_REACTIONS
        RN => REACTION(NR)
        CALL COMPUTE_RATE_CONSTANT(NR,RN%MODE,I_TS,Q_OUT,
            RATE_CONSTANT(NR),ZZ_N,I,J,K)
        IF (RATE_CONSTANT(NR) < ZERO_P) CYCLE REACTION_LOOP2
        Q_NR2(NR) = RATE_CONSTANT(NR)*RN%HEAT_OF_COMBUSTION*RHO(I
            ,J,K)
        DZZDT2 = DZZDT2 + RN%NU * SPECIES_MIXTURE%MW/
            SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW*RATE_CONSTANT(
            NR)
    END DO REACTION_LOOP2
    IF (ALL(DZZDT2 < ZERO_P)) EXIT ODE_LOOP
    ZZ_N = ZZ_I +0.5_EB*(DZZDT+DZZDT2)*DT_NEW
```

```

   IF (ANY(ZZ_N < 0._EB)) THEN
      DO NS=0,N_TRACKED_SPECIES
         IF (ZZ_N(NS) < 0._EB .AND. ABS(DZZDT(NS)+DZZDT2(NS))>
            ZERO_P) DT_NEW = MIN(DT_NEW,-2._EB*ZZ_I(NS)/(DZZDT
            (NS)+DZZDT2(NS)))
      ENDDO
   ENDIF

   Q_SUM = SUM(0.5_EB*(Q_NR+Q_NR2))

   IF (Q_OUT + Q_SUM*DT_NEW > Q_UPPER * DT) THEN
      DT_NEW = MAX(0._EB,(Q_UPPER * DT - Q_OUT))/Q_SUM
      Q_OUT = Q_OUT+Q_SUM*DT_NEW
      ZZ_I = ZZ_I + 0.5_EB*(DZZDT+DZZDT2)*DT_NEW
      EXIT ODE_LOOP
   ENDIF

   ZZ_I = ZZ_I +0.5_EB*(DZZDT+DZZDT2)*DT_NEW

   Q_OUT = Q_OUT+Q_SUM*DT_NEW

   DT_SUM = DT_SUM + DT_NEW
   IF (DT_NEW < DT_ODE) DT_NEW = DT_ODE
   IF (DT_NEW + DT_SUM > DT) DT_NEW = DT - DT_SUM
   I_TS = I_TS + 1
ENDDO ODE_LOOP

ZZ_GET = ZZ_GET + ZZ_I - ZZ_0
Q_OUT = Q_OUT / DT

RETURN

END SUBROUTINE ODE_RUNGE_KUTTA_2

SUBROUTINE ODE_RK2_RICHARDSON(I,J,K,ZZ_GET,Q_OUT)
INTEGER,INTENT(IN) :: I,J,K
REAL(EB),INTENT(OUT) :: Q_OUT
REAL(EB),INTENT(INOUT) :: ZZ_GET(0:N_TRACKED_SPECIES)
REAL(EB) :: ZZ_0(0:N_TRACKED_SPECIES),DZZDT(0:N_TRACKED_SPECIES
   ),DZZDT2(0:N_TRACKED_SPECIES),RATE_CONSTANT(1:N_REACTIONS),&
               ERR_EST,TOL_INT_VECTOR(1:N_REACTIONS),ERR_TOL,
                  Q_NR_1(1:N_REACTIONS),Q_NR2_1(1:N_REACTIONS),
```

```fortran
                    Q_NR_2(1:N_REACTIONS),&
                    Q_NR2_2(1:N_REACTIONS),Q_NR_4(1:N_REACTIONS),Q_NR2_4(1:N_REACTIONS),Q_SUM_1,Q_SUM_2,Q_SUM_4,&
                    A1(0:N_TRACKED_SPECIES),A2(0:N_TRACKED_SPECIES),A4(0:N_TRACKED_SPECIES),DT_SUB,DT_SUB_NEW,DT_ITER,&
                    DT_A1,DT_A2,DT_A4,ZZ_STORE(0:N_TRACKED_SPECIES,0:3),TV(0:2),ZZ_DIFF(0:2),Q1,Q2,Q4,Q_OUT2
INTEGER :: I_TS,NR,NS,NSS,ITER,TVI,RICH_ITER
INTEGER, PARAMETER :: NODETS=20,SUB_DT1=1,SUB_DT2=2,SUB_DT4=4,TV_ITER_MIN=5,Q_ITER_MAX=10,RICH_ITER_MAX=100
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

Q_OUT = 0._EB
Q_OUT2 = 0._EB
RICH_ITER=0
ITER=0
DT_ITER = 0._EB
I_TS = 1
DT_SUB = DT
DT_SUB_NEW = DT_SUB

! Setting up tolerance vector from inputs
DO NR = 1, N_REACTIONS
   RN => REACTION(NR)
   TOL_INT_VECTOR(NR)=RN%TOL_INT
ENDDO
ERR_TOL = MINVAL(ABS(TOL_INT_VECTOR))

INTEGRATION_LOOP: DO WHILE (DT_ITER < DT)
   ERR_EST = 10._EB*ERR_TOL
   RICH_EX_LOOP: DO WHILE (ERR_EST > ERR_TOL)
      DT_SUB = DT_SUB_NEW
      IF (DT_ITER + DT_SUB > DT) THEN
         DT_SUB = DT - DT_ITER
      ENDIF

      !------------------------
      ! Calculate A1 term
      ! Time step = DT_SUB
      !------------------------
      ZZ_0 = MAX(0._EB,ZZ_GET)
      Q1 = Q_OUT2
```

```
    ODE_LOOP1: DO NS = 1, SUB_DT1
        DZZDT = 0._EB
        DZZDT2 = 0._EB
        RATE_CONSTANT = 0._EB

        REACTION_LOOP: DO NR = 1, N_REACTIONS
            RN => REACTION(NR)
            CALL COMPUTE_RATE_CONSTANT(NR,RN%MODE,I_TS,Q_OUT2,
                RATE_CONSTANT(NR),ZZ_0,I,J,K)
            IF (RATE_CONSTANT(NR) <= 0.0_EB) CYCLE
                REACTION_LOOP
            DZZDT = DZZDT + RN%NU*SPECIES_MIXTURE%MW/
                SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW*
                RATE_CONSTANT(NR)
            Q_NR_1(NR) = RATE_CONSTANT(NR)*RN%
                HEAT_OF_COMBUSTION*RHO(I,J,K)
        END DO REACTION_LOOP
        IF (ALL(DZZDT < 0._EB)) EXIT INTEGRATION_LOOP
        A1 = ZZ_0 + DZZDT*DT_SUB
        IF (ANY(A1 < 0._EB)) THEN
            DO NSS=0,N_TRACKED_SPECIES
                IF (A1(NSS) < 0._EB .AND. ABS(DZZDT(NSS))>ZERO_P
                    ) THEN
                    DT_SUB = MIN(DT_SUB,-ZZ_0(NSS)/DZZDT(NSS))
                ENDIF
            ENDDO
            A1 = ZZ_0 + DZZDT*DT_SUB
        ENDIF

        REACTION_LOOP2: DO NR = 1, N_REACTIONS
            RN => REACTION(NR)
            CALL COMPUTE_RATE_CONSTANT(NR,RN%MODE,I_TS,Q_OUT2,
                RATE_CONSTANT(NR),A1,I,J,K)
            IF (RATE_CONSTANT(NR) <= 0.0_EB) CYCLE
                REACTION_LOOP2
            DZZDT2 = DZZDT2 + RN%NU*SPECIES_MIXTURE%MW/
                SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW*
                RATE_CONSTANT(NR)
            Q_NR2_1(NR) = RATE_CONSTANT(NR)*RN%
                HEAT_OF_COMBUSTION*RHO(I,J,K)
        END DO REACTION_LOOP2
        IF (ALL(DZZDT2 < 0._EB)) EXIT INTEGRATION_LOOP
        A1 = ZZ_0 + 0.5_EB*(DZZDT+DZZDT2)*DT_SUB
```

```
            IF (ANY(A1 < 0._EB)) THEN
               DO NSS=0,N_TRACKED_SPECIES
                  IF (A1(NSS) < 0._EB .AND. ABS(DZZDT(NSS)+DZZDT2(
                     NSS))>ZERO_P) THEN
                     DT_SUB = MIN(DT_SUB,-2._EB*ZZ_0(NSS)/(DZZDT(
                        NSS)+DZZDT2(NSS)))
                  ENDIF
               ENDDO
               A1 = ZZ_0 + 0.5_EB*(DZZDT+DZZDT2)*DT_SUB
            ENDIF

            Q_SUM_1 = SUM(0.5_EB*(Q_NR_1+Q_NR2_1))
            IF (Q1 + Q_SUM_1*DT_SUB > Q_UPPER * DT) THEN
               DT_SUB_NEW = MAX(0.0_EB,(Q_UPPER * DT-Q1)/Q_SUM_1)
               Q1 = Q1+Q_SUM_1*DT_SUB_NEW
               A1 = ZZ_0 + 0.5_EB*(DZZDT+DZZDT2)*DT_SUB_NEW
               EXIT ODE_LOOP1
            ENDIF
            Q1 = Q1+Q_SUM_1*DT_SUB
            I_TS = I_TS + 1
         ENDDO ODE_LOOP1
         DT_A1 = DT_SUB

         !------------------------
         ! Calculate A2 term
         ! Time step = DT_SUB/2
         !------------------------
         ZZ_0 = MAX(0._EB,ZZ_GET)
         Q2 = Q_OUT2
         ODE_LOOP2: DO NS = 1, SUB_DT2
            DZZDT = 0._EB
            DZZDT2 = 0._EB
            RATE_CONSTANT = 0._EB

            REACTION_LOOP_2: DO NR = 1, N_REACTIONS
               RN => REACTION(NR)
               CALL COMPUTE_RATE_CONSTANT(NR,RN%MODE,I_TS,Q_OUT2,
                  RATE_CONSTANT(NR),ZZ_0,I,J,K)
               IF (RATE_CONSTANT(NR) <= 0.0_EB) CYCLE
                  REACTION_LOOP_2
               DZZDT = DZZDT + RN%NU*SPECIES_MIXTURE%MW/
                  SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW*
                  RATE_CONSTANT(NR)
```

```
    Q_NR_2(NR) = RATE_CONSTANT(NR)*RN%HEAT_OF_COMBUSTION*RHO(I,J,K)
END DO REACTION_LOOP_2
A2 = ZZ_0 + DZZDT*(DT_SUB/REAL(SUB_DT2,EB))
IF (ANY(A2 < 0._EB)) THEN
    DO NSS=0,N_TRACKED_SPECIES
        IF (A2(NSS) < 0._EB .AND. ABS(DZZDT(NSS))>ZERO_P) THEN
            DT_SUB = MIN(DT_SUB,-ZZ_0(NSS)/DZZDT(NSS))
        ENDIF
    ENDDO
    A2 = ZZ_0 + DZZDT*(DT_SUB/REAL(SUB_DT2,EB))
ENDIF

REACTION_LOOP2_2: DO NR = 1, N_REACTIONS
    RN => REACTION(NR)
    CALL COMPUTE_RATE_CONSTANT(NR,RN%MODE,I_TS,Q_OUT2,RATE_CONSTANT(NR),A2,I,J,K)
    IF (RATE_CONSTANT(NR) <= 0.0_EB) CYCLE REACTION_LOOP2_2
    DZZDT2 = DZZDT2 + RN%NU*SPECIES_MIXTURE%MW/SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW*RATE_CONSTANT(NR)
    Q_NR2_2(NR) = RATE_CONSTANT(NR)*RN%HEAT_OF_COMBUSTION*RHO(I,J,K)
END DO REACTION_LOOP2_2
A2 = ZZ_0 + 0.5_EB*(DZZDT+DZZDT2)*(DT_SUB/REAL(SUB_DT2,EB))
IF (ANY(A2 < 0._EB)) THEN
    DO NSS=0,N_TRACKED_SPECIES
        IF (A2(NSS) < 0._EB .AND. ABS(DZZDT(NSS)+DZZDT2(NSS))>ZERO_P) THEN
            DT_SUB = MIN(DT_SUB,-2._EB*ZZ_0(NSS)/(DZZDT(NSS)+DZZDT2(NSS)))
        ENDIF
    ENDDO
    A2 = ZZ_0 + 0.5_EB*(DZZDT+DZZDT2)*(DT_SUB/REAL(SUB_DT2,EB))
ENDIF

Q_SUM_2 = SUM(0.5_EB*(Q_NR_2+Q_NR2_2))
IF (Q2+Q_SUM_2*(DT_SUB/REAL(SUB_DT2,EB)) > Q_UPPER * DT) THEN
```

```
            DT_SUB_NEW = MAX(0.0_EB,(Q_UPPER * DT-Q2)/Q_SUM_2)
            Q2 = Q2+Q_SUM_2*(DT_SUB_NEW)
            A2 = ZZ_0 + 0.5_EB*(DZZDT+DZZDT2)*(DT_SUB_NEW)
            EXIT ODE_LOOP2
         ENDIF
         Q2 = Q2+Q_SUM_2*(DT_SUB/REAL(SUB_DT2,EB))
         I_TS = I_TS + 1
         ZZ_0 = A2
      ENDDO ODE_LOOP2
      DT_A2 = DT_SUB
      IF (DT_A2 < DT_A1) THEN
         DT_SUB_NEW = DT_A2
         CYCLE RICH_EX_LOOP
      ENDIF

      !----------------------------
      ! Calculate A4 term
      ! Time step = DT_SUB/4
      !----------------------------
      ZZ_0 = MAX(0._EB,ZZ_GET)
      Q4 = Q_OUT2
      ODE_LOOP4: DO NS = 1, SUB_DT4
         DZZDT = 0._EB
         DZZDT2 = 0._EB
         RATE_CONSTANT = 0._EB

         REACTION_LOOP_4: DO NR = 1, N_REACTIONS
            RN => REACTION(NR)
            CALL COMPUTE_RATE_CONSTANT(NR,RN%MODE,I_TS,Q_OUT2,
               RATE_CONSTANT(NR),ZZ_0,I,J,K)
            IF (RATE_CONSTANT(NR) <= 0.0_EB) CYCLE
               REACTION_LOOP_4
            DZZDT = DZZDT + RN%NU*SPECIES_MIXTURE%MW/
               SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW*
               RATE_CONSTANT(NR)
            Q_NR_4(NR) = RATE_CONSTANT(NR)*RN%
               HEAT_OF_COMBUSTION*RHO(I,J,K)
         END DO REACTION_LOOP_4
         A4 = ZZ_0 + DZZDT*(DT_SUB/REAL(SUB_DT4,EB))
         IF (ANY(A4 < 0._EB)) THEN
            DO NSS=0,N_TRACKED_SPECIES
               IF (A4(NSS) < 0._EB .AND. ABS(DZZDT(NSS))>ZERO_P
                  ) THEN
                  DT_SUB = MIN(DT_SUB,-ZZ_0(NSS)/DZZDT(NSS))
               ENDIF
            ENDDO
            A4 = ZZ_0 + DZZDT*(DT_SUB/REAL(SUB_DT4,EB))
         ENDIF

         REACTION_LOOP2_4: DO NR = 1, N_REACTIONS
            RN => REACTION(NR)
            CALL COMPUTE_RATE_CONSTANT(NR,RN%MODE,I_TS,Q_OUT2,RATE_CONSTANT(NR),A4,I,J,K)
            IF (RATE_CONSTANT(NR) <= 0.0_EB) CYCLE REACTION_LOOP2_4
            DZZDT2 = DZZDT2 + RN%NU*SPECIES_MIXTURE%MW/SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW*RATE_CONSTANT(NR)
            Q_NR2_4(NR) = RATE_CONSTANT(NR)*RN%HEAT_OF_COMBUSTION*RHO(I,J,K)
         END DO REACTION_LOOP2_4
         A4 = ZZ_0 + 0.5_EB*(DZZDT+DZZDT2)*(DT_SUB/REAL(SUB_DT4,EB))
         IF (ANY(A4 < 0._EB)) THEN
            DO NSS=0,N_TRACKED_SPECIES
               IF (A4(NSS) < 0._EB .AND. ABS(DZZDT(NSS)+DZZDT2(NSS))>ZERO_P) THEN
                  DT_SUB = MIN(DT_SUB,-2._EB*ZZ_0(NSS)/(DZZDT(NSS)+DZZDT2(NSS)))
               ENDIF
            ENDDO
            A4 = ZZ_0 + 0.5_EB*(DZZDT+DZZDT2)*(DT_SUB/REAL(SUB_DT4,EB))
         ENDIF

         Q_SUM_4 = SUM(0.5_EB*(Q_NR_4+Q_NR2_4))
         IF (ABS(Q4+Q_SUM_4*(DT_SUB/REAL(SUB_DT4,EB))) > Q_UPPER*DT) THEN
            DT_SUB_NEW = MAX(0.0_EB,(Q_UPPER*DT-Q4)/Q_SUM_4)
            Q4 = Q4+Q_SUM_4*(DT_SUB_NEW)
            A4 = ZZ_0 + 0.5_EB*(DZZDT+DZZDT2)*(DT_SUB_NEW)
            IF (ITER >= Q_ITER_MAX) THEN
               Q_OUT = Q4/DT
               ZZ_GET = A4
               EXIT INTEGRATION_LOOP
            ENDIF
```

```
                EXIT ODE_LOOP4
            ENDIF
            Q4 = Q4+Q_SUM_4*(DT_SUB/REAL(SUB_DT4,EB))
            I_TS = I_TS + 1
            ZZ_0 = A4
        ENDDO ODE_LOOP4
        DT_A4 = DT_SUB
        IF (DT_A4 < DT_A2) THEN
            DT_SUB_NEW = DT_A4
            CYCLE RICH_EX_LOOP
        ENDIF

        ! Species Error Analysis
        ERR_EST = MAXVAL(ABS((4._EB*A4-A2) - (4._EB*A2-A1)))/45._EB  ! Estimate Error
        IF (ERR_EST <= 0.0_EB) THEN
            DT_SUB_NEW = DT
        ELSE
            DT_SUB_NEW =DT_SUB*(ERR_TOL/(ERR_EST))**(0.25_EB)  ! Determine New Time Step
        ENDIF

        RICH_ITER = RICH_ITER+1
        IF (RICH_ITER >= RICH_ITER_MAX) EXIT RICH_EX_LOOP
    ENDDO RICH_EX_LOOP

    DT_ITER = DT_ITER + DT_SUB
    ITER = ITER + 1
    MAX_CHEM_SUBIT = MAX(MAX_CHEM_SUBIT,ITER)
    ZZ_GET = (4._EB*A4-A2)/3._EB
    Q_OUT = (4._EB*Q4-Q2)/3._EB/DT
    Q_OUT2 = (4._EB*Q4-Q2)/3._EB

    !Total Variation Scheme
    DO NS = 0,N_TRACKED_SPECIES
        DO TVI = 0,2
            ZZ_STORE(NS,TVI)=ZZ_STORE(NS,TVI+1)
        ENDDO
        ZZ_STORE(NS,3) = (4._EB*A4(NS)-A2(NS))/3._EB
    ENDDO

    REACTION_LOOP_TV: DO NR = 1, N_REACTIONS
        RN => REACTION(NR)
```

```
      IF (.NOT. RN%REVERSIBLE) CYCLE REACTION_LOOP_TV
      DO TVI = 0,2
         TV(TVI) = ABS(ZZ_STORE(RN%FUEL_SMIX_INDEX,TVI+1)-
            ZZ_STORE(RN%FUEL_SMIX_INDEX,TVI))
         ZZ_DIFF(TVI) = ZZ_STORE(RN%FUEL_SMIX_INDEX,TVI+1)-
            ZZ_STORE(RN%FUEL_SMIX_INDEX,TVI)
      ENDDO
      IF (SUM(TV) > 0.0_EB .AND. SUM(TV) >= ABS(2.5_EB*SUM(
         ZZ_DIFF)) .AND. ITER >= TV_ITER_MIN) EXIT
         INTEGRATION_LOOP
   ENDDO REACTION_LOOP_TV

ENDDO INTEGRATION_LOOP

RETURN

END SUBROUTINE ODE_RK2_RICHARDSON

RECURSIVE SUBROUTINE COMPUTE_RATE_CONSTANT(NR,MODE,I_TS,Q_IN,
   RATE_CONSTANT,ZZ_GET,I,J,K)
USE PHYSICAL_FUNCTIONS, ONLY : GET_MASS_FRACTION_ALL,
   GET_MASS_FRACTION,GET_VISCOSITY
REAL(EB), INTENT(IN) :: ZZ_GET(0:N_TRACKED_SPECIES),Q_IN
INTEGER, INTENT(IN) :: NR,I_TS,MODE,I,J,K
REAL(EB), INTENT(INOUT) :: RATE_CONSTANT
REAL(EB) :: YY_PRIMITIVE(1:N_SPECIES),Y_F_MIN=1.E-15_EB,ZZ_MIN
   =1.E-10_EB,YY_F_LIM,ZZ_REACTANT,ZZ_PRODUCT, &
            TAU_D,TAU_G,TAU_U,DELTA,RATE_CONSTANT_ED,
               RATE_CONSTANT_FR,GAMMA_LAMBDA,NU,Y_FUEL,Y_O2,
               Y_PRODUCT,S, &
            CHI_1,CHI_2,CHI_3,CHI,C_LES  !ADDED
INTEGER :: NS
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

RN => REACTION(NR)

SELECT CASE (MODE)
   CASE(MIXED)
      IF (Q_IN > 0._EB .AND. RN%THRESHOLD_TEMP >= TMP(I,J,K))
         THEN
         CALL COMPUTE_RATE_CONSTANT(NR,EDDY_DISSIPATION,I_TS,
            Q_IN,RATE_CONSTANT,ZZ_GET,I,J,K)
      ELSE
```

```
            CALL COMPUTE_RATE_CONSTANT(NR,FINITE_RATE,I_TS,Q_IN,
                RATE_CONSTANT,ZZ_GET,I,J,K)
        ENDIF
    CASE(EDDY_DISSIPATION)
            IF_SUPPRESSION: IF (SUPPRESSION) THEN
                ! Evaluate empirical extinction criteria
                IF (I_TS==1) THEN
                    IF(EXTINCTION(I,J,K,ZZ_GET)) THEN
                        RATE_CONSTANT = 0._EB
                        RETURN
                    ENDIF
                !ELSE
                !    IF (RATE_CONSTANT <= ZERO_P) RETURN
                ENDIF
            ENDIF IF_SUPPRESSION

            FIXED_TIME: IF (FIXED_MIX_TIME>0._EB) THEN
                MIX_TIME(I,J,K)=FIXED_MIX_TIME

            ELSE FIXED_TIME
                IF (TWO_D) THEN
                    DELTA = MAX(DX(I),DZ(K))
                ELSE
                    DELTA = MAX(DX(I),DY(J),DZ(K))
                ENDIF

                !LES_IF: IF (LES) THEN

                    !TAU_D = D_Z(MIN(4999,NINT(TMP(I,J,K))),RN%
                        FUEL_SMIX_INDEX)
                    !TAU_D = DELTA**2/TAU_D ! diffusive time scale

                    !IF (TURB_MODEL==DEARDORFF) THEN
                        !TAU_U = 0.1_EB*SC*RHO(I,J,K)*DELTA**2/MU(I,J
                            ,K) ! turbulent mixing time scale
                    !ELSE
                        !TAU_U = DELTA/SQRT(2._EB*KSGS(I,J,K)+1.E-10
                            _EB) ! advective time scale
                    !ENDIF

                    !TAU_G = SQRT(2._EB*DELTA/(GRAV+1.E-10_EB)) !
                        acceleration time scale

            !MIX_TIME(I,J,K)=MAX(TAU_CHEM,MIN(TAU_D,TAU_U,TAU_G,TAU_FLAME)) ! Eq. 7, McDermott, McGrattan, Floyd

         !ELSE LES_IF

            !TAU_D = D_Z(MIN(4999,NINT(TMP(I,J,K))),RN%FUEL_SMIX_INDEX)
            !TAU_D = DELTA**2/TAU_D
            !MIX_TIME(I,J,K)= TAU_D

         !ENDIF LES_IF
         MIX_TIME(I,J,K) = 1._EB/ABS(STRAIN_RATE(I,J,K)) ! ADDED
      ENDIF FIXED_TIME

      YY_F_LIM=1.E15_EB
      IF (N_REACTIONS > 1) THEN
         DO NS=0,N_TRACKED_SPECIES
            IF(RN%NU(NS) < -ZERO_P) THEN
               IF (ZZ_GET(NS) < ZZ_MIN) THEN
                  RATE_CONSTANT = 0._EB
                  RETURN
               ENDIF
               YY_F_LIM = MIN(YY_F_LIM,&
                                 ZZ_GET(NS)*SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW/(ABS(RN%NU(NS))*SPECIES_MIXTURE(NS)%MW))
            ENDIF
         ENDDO
      ELSE
         ZZ_REACTANT = 0._EB
         ZZ_PRODUCT = 0._EB
         DO NS=0,N_TRACKED_SPECIES
            IF(RN%NU(NS) < -ZERO_P) THEN
               IF (ZZ_GET(NS) < ZZ_MIN) THEN
                  RATE_CONSTANT = 0._EB
                  RETURN
               ENDIF
               ZZ_REACTANT = ZZ_REACTANT - RN%NU(NS)*SPECIES_MIXTURE(NS)%MW
               YY_F_LIM = MIN(YY_F_LIM,&
                                                   ZZ_GET(NS)*SPECIES_MIXTURE(RN%
                                                       FUEL_SMIX_INDEX)%MW/(ABS(RN
                                                       %NU(NS))*SPECIES_MIXTURE(NS
                                                       )%MW))
                  ELSEIF(RN%NU(NS)>ZERO_P ) THEN
                      ZZ_PRODUCT = ZZ_PRODUCT + ZZ_GET(NS)
                  ENDIF
              ENDDO
              ZZ_PRODUCT = BETA_EDC*MAX(ZZ_PRODUCT*
                  SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW/
                  ZZ_REACTANT,Y_P_MIN_EDC)
              YY_F_LIM = MIN(YY_F_LIM,ZZ_PRODUCT)
          ENDIF
          YY_F_LIM = MAX(YY_F_LIM,Y_F_MIN)
          !RATE_CONSTANT = YY_F_LIM/MIX_TIME(I,J,K)

          !The Eddy Dissipation Consept (EDC) Combustion Model (
              by Hjertager and Magnussen) for LES proposed by
              Balram et al.
          NU = MU(I,J,K)/RHO(I,J,K)
          C_LES = 0.10_EB
          !IF(SELECT_TURB == DEARDORFF) THEN
          !      C_LES =
          !ELSEIF(SELECT_TURB == DYNSMAG)THEN
          !      C_LES =
          !ELSEIF(SELECT_TURB == VREMAN)THEN
          !      C_LES =
          !ELSE
          !      WRITE(*,*) 'The chosen turbulence model is not
              supported by the combustion model'
          !END

          GAMMA_LAMBDA = C_LES*(NU/NU_EDDY(I,J,K))**0.25_EB
          IF(GAMMA_LAMBDA > 1._EB)THEN
              GAMMA_LAMBDA = 1._EB
          END IF

          CALL GET_MASS_FRACTION(ZZ_GET,FUEL_INDEX,Y_FUEL) !
              ADDED
          CALL GET_MASS_FRACTION(ZZ_GET,O2_INDEX,Y_O2) !ADDED
          Y_PRODUCT = 1._EB - (Y_FUEL + Y_O2)

```

```
            S = SPECIES(FUEL_INDEX)%MW/(RN%NU_O2*SPECIES(O2_INDEX)
                %MW)
            Y_O2 = Y_O2/S
            Y_PRODUCT = Y_PRODUCT/(1._EB + S)
            CHI_1 = ((YY_F_LIM + Y_PRODUCT)**2)/((Y_FUEL +
                Y_PRODUCT)*(Y_O2 + Y_PRODUCT))
            CHI_2 = MIN(Y_PRODUCT/(GAMMA_LAMBDA*(YY_F_LIM +
                Y_PRODUCT)),1._EB)
            CHI_3 = MIN(GAMMA_LAMBDA*(YY_F_LIM + Y_PRODUCT)/
                YY_F_LIM,1._EB)
            CHI = CHI_1*CHI_2*CHI_3

            RATE_CONSTANT = YY_F_LIM*CHI/(MIX_TIME(I,J,K)*(1._EB -
                 CHI*GAMMA_LAMBDA**2))
            !RATE_CONSTANT = YY_F_LIM*CHI*GAMMA_LAMBDA**2/(
                MIX_TIME(I,J,K)*(1._EB - CHI*GAMMA_LAMBDA**2))


    CASE(FINITE_RATE)
        RATE_CONSTANT = 0._EB
        CALL GET_MASS_FRACTION_ALL(ZZ_GET,YY_PRIMITIVE)
        RATE_CONSTANT = RN%A*RHO(I,J,K)**RN%RHO_EXPONENT*EXP(-RN%
            E/(R0*TMP(I,J,K)))*TMP(I,J,K)**RN%N_T
        IF (ALL(RN%N_S<-998._EB)) THEN
            DO NS=0,N_TRACKED_SPECIES
                IF(RN%NU(NS)<0._EB .AND. ZZ_GET(NS) < ZZ_MIN) THEN
                    RATE_CONSTANT = 0._EB
                    RETURN
                ENDIF
            ENDDO
        ELSE
            DO NS=1,N_SPECIES
                IF(ABS(RN%N_S(NS)) <= ZERO_P) CYCLE
                IF(RN%N_S(NS)>= -998._EB) THEN
                    IF (YY_PRIMITIVE(NS) < ZZ_MIN) THEN
                        RATE_CONSTANT = 0._EB
                    ELSE
                        RATE_CONSTANT = YY_PRIMITIVE(NS)**RN%N_S(NS)*
                            RATE_CONSTANT
                    ENDIF
                ENDIF
            ENDDO
        ENDIF
```

```

   CASE(EDDY_DISSIPATION_CONCEPT)
      CALL COMPUTE_RATE_CONSTANT(NR,EDDY_DISSIPATION,I_TS,Q_IN,
         RATE_CONSTANT,ZZ_GET,I,J,K)
      RATE_CONSTANT_ED=RATE_CONSTANT
      CALL COMPUTE_RATE_CONSTANT(NR,FINITE_RATE,I_TS,Q_IN,
         RATE_CONSTANT,ZZ_GET,I,J,K)
      RATE_CONSTANT_FR=RATE_CONSTANT
      IF (ABS(RATE_CONSTANT_ED) < ZERO_P .AND. ABS(
         RATE_CONSTANT_FR) < ZERO_P) THEN
         RATE_CONSTANT=0.0_EB
      ELSE
         RATE_CONSTANT = (RATE_CONSTANT_ED*RATE_CONSTANT_FR)/(
            RATE_CONSTANT_ED+RATE_CONSTANT_FR)
      ENDIF
END SELECT

RETURN

CONTAINS

LOGICAL FUNCTION EXTINCTION(I,J,K,ZZ_IN)
!This routine determines if local extinction occurs for a
    mixing controlled reaction.
!This is determined as follows:
!1) Determine how much fuel can burn (DZ_FUEL) by finding the
    limiting reactant and expressing it in terms of fuel mass
!2) Remove that amount of fuel form the local mixture,
    everything else is "air"
!   (i.e. if we are fuel rich, excess fuel acts as a diluent)
!3) Search to find the minimum reactant other than fuel.
!   Using the reaction stoichiometry, determine how much "air"
    (DZ_AIR) is needed to burn the fuel.
!4) GET_AVERAGE_SPECIFIC_HEAT for the fuel and the "air" at the
     current temp and the critical flame temp
!5) Check to see if the heat released from burning DZ_FUEL can
    raise the current temperature of DZ_FUEL and DZ_AIR
!   above the critical flame temp.
USE PHYSICAL_FUNCTIONS,ONLY:GET_AVERAGE_SPECIFIC_HEAT
REAL(EB),INTENT(IN)::ZZ_IN(0:N_TRACKED_SPECIES)
REAL(EB):: DZ_AIR,DZ_FUEL,CPBAR_F_0,CPBAR_F_N,CPBAR_G_0,
    CPBAR_G_N,ZZ_GET(0:N_TRACKED_SPECIES)
INTEGER, INTENT(IN) :: I,J,K
```

```
INTEGER :: NS

EXTINCTION = .FALSE.
IF (TMP(I,J,K) < RN%AUTO_IGNITION_TEMPERATURE) THEN
    EXTINCTION = .TRUE.
ELSE
    DZ_FUEL = 1._EB
    DZ_AIR = 0._EB
    !Search reactants to find limiting reactant and express it
        as fuel mass.  This is the amount of fuel
    !that can burn
    DO NS = 0,N_TRACKED_SPECIES
        IF (RN%NU(NS)<-ZERO_P) &
            DZ_FUEL = MIN(DZ_FUEL,-ZZ_IN(NS)*SPECIES_MIXTURE(RN%
                FUEL_SMIX_INDEX)%MW/(RN%NU(NS)*SPECIES_MIXTURE(NS)%
                MW))
    ENDDO
    !Get the specific heat for the fuel at the current and
        critical flame temperatures
    ZZ_GET = 0._EB
    ZZ_GET(RN%FUEL_SMIX_INDEX) = 1._EB
    CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ_GET,CPBAR_F_0,TMP(I,J,K))
    CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ_GET,CPBAR_F_N,RN%
        CRIT_FLAME_TMP)
    ZZ_GET = ZZ_IN
    !Remove the burnable fuel from the local mixture and
        renormalize.  The remainder is "air"
    ZZ_GET(RN%FUEL_SMIX_INDEX) = ZZ_GET(RN%FUEL_SMIX_INDEX) -
        DZ_FUEL
    ZZ_GET = ZZ_GET/SUM(ZZ_GET)
    !Get the specific heat for the "air"
    CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ_GET,CPBAR_G_0,TMP(I,J,K))
    CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ_GET,CPBAR_G_N,RN%
        CRIT_FLAME_TMP)
    !Loop over non-fuel reactants and find the mininum.
        Determine how much "air" is needed to provide the limting
         reactant
    DO NS = 0,N_TRACKED_SPECIES
            IF (RN%NU(NS)<-ZERO_P .AND. NS/=RN%FUEL_SMIX_INDEX)
                &
              DZ_AIR = MAX(DZ_AIR, -DZ_FUEL*RN%NU(NS)*
                  SPECIES_MIXTURE(NS)%MW/SPECIES_MIXTURE(RN%
                  FUEL_SMIX_INDEX)%MW/ZZ_GET(NS))
    ENDDO
    !See if enough energy is released to raise the fuel and
        required "air" temperatures above the critical flame temp
    IF ( (DZ_FUEL*CPBAR_F_0 + DZ_AIR*CPBAR_G_0)*TMP(I,J,K) +
        DZ_FUEL*RN%HEAT_OF_COMBUSTION < &
            (DZ_FUEL*CPBAR_F_N + DZ_AIR*CPBAR_G_N)*RN%
                CRIT_FLAME_TMP) EXTINCTION = .TRUE.
ENDIF

END FUNCTION EXTINCTION


REAL(EB) FUNCTION KSGS(I,J,K)
INTEGER, INTENT(IN) :: I,J,K
REAL(EB) :: EPSK

! ke dissipation rate, assumes production=dissipation

EPSK = MU(I,J,K)*STRAIN_RATE(I,J,K)**2/RHO(I,J,K)

KSGS = 2.25_EB*(EPSK*DELTA/PI)**TWTH  ! estimate of subgrid ke,
      from Kolmogorov spectrum

END FUNCTION KSGS

END SUBROUTINE COMPUTE_RATE_CONSTANT


SUBROUTINE GET_REV_fire(MODULE_REV,MODULE_DATE)
INTEGER,INTENT(INOUT) :: MODULE_REV
CHARACTER(255),INTENT(INOUT) :: MODULE_DATE

WRITE(MODULE_DATE,'(A)') firerev(INDEX(firerev,':')+2:LEN_TRIM(
      firerev)-2)
READ (MODULE_DATE,'(I5)') MODULE_REV
WRITE(MODULE_DATE,'(A)') firedate

END SUBROUTINE GET_REV_fire

END MODULE FIRE
```

B.2 velo.f90

Modified subroutine in velo.f90:

```
SUBROUTINE COMPUTE_VISCOSITY(NM)

USE PHYSICAL_FUNCTIONS, ONLY: GET_VISCOSITY
USE TURBULENCE, ONLY: VARDEN_DYNSMAG,TEST_FILTER,EX2G3D
INTEGER, INTENT(IN) :: NM
REAL(EB) :: ZZ_GET(0:N_TRACKED_SPECIES),DELTA,NU_G,GRAD_RHO(3),
    U2,V2,W2,AA,A_IJ(3,3),BB,B_IJ(3,3),&
                 DUDX,DUDY,DUDZ,DVDX,DVDY,DVDZ,DWDX,DWDY,DWDZ,MU_DNS
                    ,KSGS !,NU_EDDY(1:IBAR,1:JBAR,1:KBAR)
INTEGER :: I,J,K,IIG,JJG,KKG,II,JJ,KK,IW,TURB_MODEL_TMP,IOR
REAL(EB), POINTER, DIMENSION(:,:,:) :: RHOP=>NULL(),UP=>NULL(),
    VP=>NULL(),WP=>NULL(), &
                                                   UP_HAT=>NULL(),VP_HAT=>
                                                       NULL(),WP_HAT=>NULL()
                                                       , &
                                                   UU=>NULL(),VV=>NULL(),WW
                                                       =>NULL()
REAL(EB), POINTER, DIMENSION(:,:,:,:) :: ZZP=>NULL()
TYPE(WALL_TYPE), POINTER :: WC=>NULL()

CALL POINT_TO_MESH(NM)

IF (PREDICTOR) THEN
    RHOP => RHO
    UU   => U
    VV   => V
    WW   => W
    IF (N_TRACKED_SPECIES > 0) ZZP => ZZ
ELSE
    RHOP => RHOS
    UU   => US
    VV   => VS
    WW   => WS
    IF (N_TRACKED_SPECIES > 0 .AND. .NOT.EVACUATION_ONLY(NM))
        ZZP => ZZS
ENDIF

! Compute viscosity for DNS using primitive species/mixture
    fraction

```

```
!$OMP PARALLEL DEFAULT(NONE) &
!$OMP SHARED(N_TRACKED_SPECIES,EVACUATION_ONLY,KBAR,JBAR,IBAR,
    SOLID,CELL_INDEX,ZZP,MU,TMP, &
!$OMP            LES,NM,C_SMAGORINSKY,TWO_D,DX,DY,DZ,RDX,RDY,RDZ,UU
    ,VV,WW,RHOP,CSD2, &
!$OMP            N_EXTERNAL_WALL_CELLS,N_INTERNAL_WALL_CELLS,KRES,
    &
!$OMP            IBP1,JBP1,KBP1,TURB_MODEL_TMP,TURB_MODEL,PREDICTOR
    ,STRAIN_RATE,UP,VP,WP,WORK1,WORK2,WORK3,WC,WALL,U_GHOST,
    V_GHOST, &
!$OMP            W_GHOST,UP_HAT,VP_HAT,WP_HAT,WORK4,WORK5,WORK6,
    DELTA,KSGS,NU_EDDY,C_DEARDORFF,DUDX,DVDY,DWDZ,DUDY,DUDZ,DVDX
    ,DVDZ, &
!$OMP            DWDX,DWDY,II,JJ,KK,A_IJ,AA,B_IJ,BB,C_VREMAN,
    GRAV_VISC,GRAD_RHO,NU_G,C_G,GVEC,IOR,MU_DNS) &
!$OMP PRIVATE(ZZ_GET)

IF (N_TRACKED_SPECIES>0 .AND. EVACUATION_ONLY(NM)) ZZ_GET(1:
    N_TRACKED_SPECIES) = 0._EB

!$OMP DO COLLAPSE(3) SCHEDULE(STATIC) &
!$OMP PRIVATE(K,J,I)
DO K=1,KBAR
   DO J=1,JBAR
      DO I=1,IBAR
         IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
         IF (N_TRACKED_SPECIES>0 .AND. .NOT.EVACUATION_ONLY(NM)
            ) ZZ_GET(1:N_TRACKED_SPECIES) = ZZP(I,J,K,1:
            N_TRACKED_SPECIES)
         CALL GET_VISCOSITY(ZZ_GET,MU(I,J,K),TMP(I,J,K))
      ENDDO
   ENDDO
ENDDO
!$OMP END DO


TURB_MODEL_TMP = TURB_MODEL
IF (EVACUATION_ONLY(NM)) TURB_MODEL_TMP = CONSMAG

SELECT_TURB: SELECT CASE (TURB_MODEL_TMP)

   CASE (CONSMAG,DYNSMAG) SELECT_TURB ! Smagorinsky (1963) eddy
        viscosity

      CALL COMPUTE_STRAIN_RATE(NM)

      IF (PREDICTOR .AND. TURB_MODEL_TMP==DYNSMAG) CALL
          VARDEN_DYNSMAG(NM) ! dynamic procedure, Moin et al.
          (1991)

      !$OMP DO COLLAPSE(3) SCHEDULE(STATIC) &
      !$OMP PRIVATE(K,J,I)
      DO K=1,KBAR
         DO J=1,JBAR
            DO I=1,IBAR
               IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
               MU(I,J,K) = MU(I,J,K) + RHOP(I,J,K)*CSD2(I,J,K)*
                  STRAIN_RATE(I,J,K)
               !NU_EDDY = MU(I,J,K)/RHO(I,J,K)
               NU_EDDY(I,J,K) = MU(I,J,K)/RHO(I,J,K)
            ENDDO
         ENDDO
      ENDDO
      !$OMP END DO

   CASE (DEARDORFF) SELECT_TURB ! Deardorff (1980) eddy
      viscosity model (current default)

      ! Velocities relative to the p-cell center

!!!        CALL COMPUTE_STRAIN_RATE(NM) !til forbrenningsmodell

      UP => WORK1
      VP => WORK2
      WP => WORK3
      UP=0._EB
      VP=0._EB
      WP=0._EB

      DO K=1,KBAR
         DO J=1,JBAR
            DO I=1,IBAR
               UP(I,J,K) = 0.5_EB*(UU(I,J,K) + UU(I-1,J,K))
               VP(I,J,K) = 0.5_EB*(VV(I,J,K) + VV(I,J-1,K))
               WP(I,J,K) = 0.5_EB*(WW(I,J,K) + WW(I,J,K-1))
            ENDDO
         ENDDO
      ENDDO

      ! extrapolate to ghost cells

      CALL EX2G3D(UP,-1.E10_EB,1.E10_EB)
      CALL EX2G3D(VP,-1.E10_EB,1.E10_EB)
      CALL EX2G3D(WP,-1.E10_EB,1.E10_EB)

      DO IW=1,N_EXTERNAL_WALL_CELLS
         WC=>WALL(IW)
         IF (WC%BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY) CYCLE
          II = WC%II
          JJ = WC%JJ
          KK = WC%KK
          UP(II,JJ,KK) = U_GHOST(IW)
          VP(II,JJ,KK) = V_GHOST(IW)
          WP(II,JJ,KK) = W_GHOST(IW)
      ENDDO

      UP_HAT => WORK4
      VP_HAT => WORK5
      WP_HAT => WORK6
      UP_HAT=0._EB
      VP_HAT=0._EB
      WP_HAT=0._EB

      CALL TEST_FILTER(UP_HAT,UP,-1.E10_EB,1.E10_EB)
      CALL TEST_FILTER(VP_HAT,VP,-1.E10_EB,1.E10_EB)
      CALL TEST_FILTER(WP_HAT,WP,-1.E10_EB,1.E10_EB)

      DO K=1,KBAR
          DO J=1,JBAR
            DO I=1,IBAR
               IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
               IF (TWO_D) THEN
                  DELTA = MAX(DX(I),DZ(K))
               ELSE
                  DELTA = MAX(DX(I),DY(J),DZ(K))
               ENDIF

               KSGS = 0.5_EB*( (UP(I,J,K)-UP_HAT(I,J,K))**2 + (
                  VP(I,J,K)-VP_HAT(I,J,K))**2 + (WP(I,J,K)-
```

```
                    WP_HAT(I,J,K))**2  )
               !KSGS(I,J,K) = 0.5_EB*(  (UP(I,J,K)-UP_HAT(I,J,K)
                    )**2 + (VP(I,J,K)-VP_HAT(I,J,K))**2 + (WP(I,J
                    ,K)-WP_HAT(I,J,K))**2  )
               !NU_EDDY(I,J,K) = C_DEARDORFF*DELTA*SQRT(KSGS)
               NU_EDDY(I,J,K) = C_DEARDORFF*DELTA*SQRT(KSGS)

               MU(I,J,K) = MU(I,J,K) + RHOP(I,J,K)*NU_EDDY(I,J,
                    K)
            ENDDO
         ENDDO
      ENDDO

   CASE (VREMAN) SELECT_TURB ! Vreman (2004) eddy viscosity
         model (experimental)

      ! A. W. Vreman. An eddy-viscosity subgrid-scale model for
             turbulent shear flow: Algebraic theory and
             applications.
      ! Phys. Fluids, 16(10):3670-3681, 2004.

      DO K=1,KBAR
         DO J=1,JBAR
            DO I=1,IBAR
               IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
               DUDX = RDX(I)*(UU(I,J,K)-UU(I-1,J,K))
               DVDY = RDY(J)*(VV(I,J,K)-VV(I,J-1,K))
               DWDZ = RDZ(K)*(WW(I,J,K)-WW(I,J,K-1))
               DUDY = 0.25_EB*RDY(J)*(UU(I,J+1,K)-UU(I,J-1,K)+
                    UU(I-1,J+1,K)-UU(I-1,J-1,K))
               DUDZ = 0.25_EB*RDZ(K)*(UU(I,J,K+1)-UU(I,J,K-1)+
                    UU(I-1,J,K+1)-UU(I-1,J,K-1))
               DVDX = 0.25_EB*RDX(I)*(VV(I+1,J,K)-VV(I-1,J,K)+
                    VV(I+1,J-1,K)-VV(I-1,J-1,K))
               DVDZ = 0.25_EB*RDZ(K)*(VV(I,J,K+1)-VV(I,J,K-1)+
                    VV(I,J-1,K+1)-VV(I,J-1,K-1))
               DWDX = 0.25_EB*RDX(I)*(WW(I+1,J,K)-WW(I-1,J,K)+
                    WW(I+1,J,K-1)-WW(I-1,J,K-1))
               DWDY = 0.25_EB*RDY(J)*(WW(I,J+1,K)-WW(I,J-1,K)+
                    WW(I,J+1,K-1)-WW(I,J-1,K-1))

               ! Vreman, Eq. (6)
               A_IJ(1,1)=DUDX; A_IJ(2,1)=DUDY; A_IJ(3,1)=DUDZ
         A_IJ(1,2)=DVDX;  A_IJ(2,2)=DVDY;  A_IJ(3,2)=DVDZ
         A_IJ(1,3)=DWDX;  A_IJ(2,3)=DWDY;  A_IJ(3,3)=DWDZ

        AA=0._EB
        DO JJ=1,3
           DO II=1,3
              AA = AA + A_IJ(II,JJ)*A_IJ(II,JJ)
           ENDDO
        ENDDO

        ! Vreman, Eq. (7)
         B_IJ(1,1)=(DX(I)*A_IJ(1,1))**2 + (DY(J)*A_IJ
            (2,1))**2 + (DZ(K)*A_IJ(3,1))**2
         B_IJ(2,2)=(DX(I)*A_IJ(1,2))**2 + (DY(J)*A_IJ
            (2,2))**2 + (DZ(K)*A_IJ(3,2))**2
         B_IJ(3,3)=(DX(I)*A_IJ(1,3))**2 + (DY(J)*A_IJ
            (2,3))**2 + (DZ(K)*A_IJ(3,3))**2

         B_IJ(1,2)=DX(I)**2*A_IJ(1,1)*A_IJ(1,2) + DY(J)
            **2*A_IJ(2,1)*A_IJ(2,2) + DZ(K)**2*A_IJ(3,1)*
            A_IJ(3,2)
         B_IJ(1,3)=DX(I)**2*A_IJ(1,1)*A_IJ(1,3) + DY(J)
            **2*A_IJ(2,1)*A_IJ(2,3) + DZ(K)**2*A_IJ(3,1)*
            A_IJ(3,3)
         B_IJ(2,3)=DX(I)**2*A_IJ(1,2)*A_IJ(1,3) + DY(J)
            **2*A_IJ(2,2)*A_IJ(2,3) + DZ(K)**2*A_IJ(3,2)*
            A_IJ(3,3)

        BB = B_IJ(1,1)*B_IJ(2,2) - B_IJ(1,2)**2 &
           + B_IJ(1,1)*B_IJ(3,3) - B_IJ(1,3)**2 &
           + B_IJ(2,2)*B_IJ(3,3) - B_IJ(2,3)**2      !
              Vreman, Eq. (8)

         IF (ABS(AA)>ZERO_P) THEN
            NU_EDDY(I,J,K) = C_VREMAN*SQRT(BB/AA)   !
              Vreman, Eq. (5)
        ELSE
            NU_EDDY(I,J,K)=0._EB
        ENDIF

        MU(I,J,K) = MU(I,J,K) + RHOP(I,J,K)*NU_EDDY(I,J,
           K)

```

```
                    ENDDO
                ENDDO
            ENDDO

END SELECT SELECT_TURB

! Add viscosity for stably stratified flows (experimental)

GRAVITY_IF: IF (LES .AND. GRAV_VISC) THEN

    DO K=1,KBAR
        DO J=1,JBAR
            DO I=1,IBAR
                IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
                IF (TWO_D) THEN
                    DELTA = MAX(DX(I),DZ(K))
                ELSE
                    DELTA = MAX(DX(I),DY(J),DZ(K))
                ENDIF

                GRAD_RHO(1) = 0.5_EB*RDX(I)*(RHOP(I+1,J,K)-RHOP(I
                    -1,J,K))
                GRAD_RHO(2) = 0.5_EB*RDY(J)*(RHOP(I,J+1,K)-RHOP(I,J
                    -1,K))
                GRAD_RHO(3) = 0.5_EB*RDZ(K)*(RHOP(I,J,K+1)-RHOP(I,J
                    ,K-1))

                NU_G = C_G*DELTA**2*SQRT(MAX(ZERO_P,DOT_PRODUCT(
                    GRAD_RHO,GVEC))/RHOP(I,J,K))

                MU(I,J,K) = MAX(MU(I,J,K),RHOP(I,J,K)*NU_G)
            ENDDO
        ENDDO
    ENDDO

ENDIF GRAVITY_IF

! Compute resolved kinetic energy per unit mass

!$OMP DO COLLAPSE(3) SCHEDULE(STATIC) PRIVATE(K,J,I,U2,V2,W2)
DO K=1,KBAR
    DO J=1,JBAR
        DO I=1,IBAR
```

```
            U2 = 0.25_EB*(UU(I-1,J,K)+UU(I,J,K))**2
            V2 = 0.25_EB*(VV(I,J-1,K)+VV(I,J,K))**2
            W2 = 0.25_EB*(WW(I,J,K-1)+WW(I,J,K))**2
            KRES(I,J,K) = 0.5_EB*(U2+V2+W2)
         ENDDO
      ENDDO
   ENDDO
   !$OMP END DO NOWAIT

   ! Mirror viscosity into solids and exterior boundary cells

   !$OMP DO SCHEDULE(STATIC) &
   !$OMP PRIVATE(IW, II, JJ, KK, IIG, JJG, KKG)
   WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS
      WC=>WALL(IW)
      IF (WC%BOUNDARY_TYPE==NULL_BOUNDARY) CYCLE WALL_LOOP
      II  = WC%II
      JJ  = WC%JJ
      KK  = WC%KK
      IOR = WC%IOR
      IIG = WC%IIG
      JJG = WC%JJG
      KKG = WC%KKG

      SELECT CASE(WC%BOUNDARY_TYPE)
         CASE(SOLID_BOUNDARY)
            IF (LES) THEN
               IF (N_TRACKED_SPECIES>0 .AND. .NOT.EVACUATION_ONLY(
                  NM)) &
                  ZZ_GET(1:N_TRACKED_SPECIES) = ZZP(IIG,JJG,KKG,1:
                     N_TRACKED_SPECIES)
               CALL GET_VISCOSITY(ZZ_GET,MU_DNS,TMP(IIG,JJG,KKG))
               SELECT CASE (IOR)
                  CASE ( 1); MU(IIG,JJG,KKG) = MAX(MU_DNS,ONTH*MU(
                     IIG+1,JJG,KKG))
                  CASE (-1); MU(IIG,JJG,KKG) = MAX(MU_DNS,ONTH*MU(
                     IIG-1,JJG,KKG))
                  CASE ( 2); MU(IIG,JJG,KKG) = MAX(MU_DNS,ONTH*MU(
                     IIG,JJG+1,KKG))
                  CASE (-2); MU(IIG,JJG,KKG) = MAX(MU_DNS,ONTH*MU(
                     IIG,JJG-1,KKG))
                  CASE ( 3); MU(IIG,JJG,KKG) = MAX(MU_DNS,ONTH*MU(
                     IIG,JJG,KKG+1))
```

```
                    CASE (-3); MU(IIG,JJG,KKG) = MAX(MU_DNS,ONTH*MU(
                        IIG,JJG,KKG-1))
                 END SELECT
              ENDIF
              IF (SOLID(CELL_INDEX(II,JJ,KK))) MU(II,JJ,KK) = MU(IIG
                 ,JJG,KKG)
         CASE(OPEN_BOUNDARY,MIRROR_BOUNDARY)
              MU(II,JJ,KK) = MU(IIG,JJG,KKG)
              KRES(II,JJ,KK) = KRES(IIG,JJG,KKG)
     END SELECT
ENDDO WALL_LOOP
!$OMP END DO

!$OMP WORKSHARE
MU(     0,0:JBP1,     0) = MU(     1,0:JBP1,1)
MU(IBP1,0:JBP1,     0) = MU(IBAR,0:JBP1,1)
MU(IBP1,0:JBP1,KBP1) = MU(IBAR,0:JBP1,KBAR)
MU(     0,0:JBP1,KBP1) = MU(     1,0:JBP1,KBAR)
MU(0:IBP1,     0,     0) = MU(0:IBP1,     1,1)
MU(0:IBP1,JBP1,0)      = MU(0:IBP1,JBAR,1)
MU(0:IBP1,JBP1,KBP1) = MU(0:IBP1,JBAR,KBAR)
MU(0:IBP1,0,KBP1)      = MU(0:IBP1,     1,KBAR)
MU(0,     0,0:KBP1)      = MU(     1,     1,0:KBP1)
MU(IBP1,0,0:KBP1)      = MU(IBAR,     1,0:KBP1)
MU(IBP1,JBP1,0:KBP1) = MU(IBAR,JBAR,0:KBP1)
MU(0,JBP1,0:KBP1)      = MU(     1,JBAR,0:KBP1)
!$OMP END WORKSHARE
!$OMP END PARALLEL

END SUBROUTINE COMPUTE_VISCOSITY
```

B.3 mesh.f90

Modified subroutine in mesh.f90:

```
MODULE MESH_VARIABLES

! Data structure for mesh-dependent variables

USE PRECISION_PARAMETERS
USE TYPES
IMPLICIT NONE

CHARACTER(255), PARAMETER :: meshid='$Id:␣mesh.f90␣10087␣2012-02-15␣21:06:17Z␣randy.mcdermott␣$'
CHARACTER(255), PARAMETER :: meshrev='$Revision:␣10087␣$'
CHARACTER(255), PARAMETER :: meshdate='$Date:␣2012-02-15␣22:06:17␣+0100␣(on,␣15␣feb␣2012)␣$'

TYPE MESH_TYPE

   REAL(EB), POINTER, DIMENSION(:,:,:) :: &
             U,V,W,US,VS,WS,DDDT,D,DS,H,HS,KRES,FVX,FVY,FVZ,RHO,RHOS, &
             MU,TMP,Q,FRHO,KAPPA,QR,QR_W,UII,RSUM,D_LAGRANGIAN,D_REACTION, &
             CSD2,MIX_TIME,STRAIN_RATE,KFST4,RHO_H_S_OVER_PBAR,D_RHSOP_DT,D_RHSOP_DT_S,NU_EDDY !ADDED

   REAL(EB), POINTER, DIMENSION(:,:,:,:) :: ZZ,ZZS,DEL_RHO_D_DEL_Z
   REAL(EB), POINTER, DIMENSION(:,:,:,:) :: AVG_DROP_DEN,AVG_DROP_TMP,AVG_DROP_RAD,AVG_DROP_AREA
   REAL(EB), POINTER, DIMENSION(:,:,:) :: AVG_DROP_DEN_ALL
   REAL(EB), POINTER, DIMENSION(:,:) :: UVW_GHOST

   REAL(EB) :: POIS_PTB,POIS_ERR
   REAL(EB), POINTER, DIMENSION(:) :: SAVE1,SAVE2,WORK
   REAL(EB), POINTER, DIMENSION(:,:,:) :: PRHS
   REAL(EB), POINTER, DIMENSION(:,:) :: BXS,BXF,BYS,BYF,BZS,BZF, BXST,BXFT,BYST,BYFT,BZST,BZFT
   INTEGER :: LSAVE,LWORK,LBC,MBC,NBC,ITRN,JTRN,KTRN,IPS
   REAL(EB), POINTER, DIMENSION(:) :: P_0,RHO_0,TMP_0,D_PBAR_DT,D_PBAR_S_DT,U_LEAK,U_DUCT
   REAL(EB), POINTER, DIMENSION(:,:) :: PBAR,PBAR_S,R_PBAR
   INTEGER, POINTER, DIMENSION(:,:,:) :: PRESSURE_ZONE
   INTEGER, POINTER, DIMENSION(:) :: PRESSURE_BC_INDEX
   REAL(EB), POINTER, DIMENSION(:,:,:) :: WORK1,WORK2,WORK3,WORK4,WORK5,WORK6,WORK7,WORK8

   REAL(EB), POINTER, DIMENSION(:,:,:) :: TURB_WORK1,TURB_WORK2,TURB_WORK3,TURB_WORK4
   REAL(EB), POINTER, DIMENSION(:,:,:) :: TURB_WORK5,TURB_WORK6,TURB_WORK7,TURB_WORK8
   REAL(EB), POINTER, DIMENSION(:,:,:) :: TURB_WORK9,TURB_WORK10
   REAL(EB), POINTER, DIMENSION(:) :: TURB_WORK11,TURB_WORK12

   REAL(EB), POINTER, DIMENSION(:,:,:) :: IBM_SAVE1,IBM_SAVE2,IBM_SAVE3,IBM_SAVE4,IBM_SAVE5,IBM_SAVE6
   INTEGER, POINTER, DIMENSION(:,:,:) :: U_MASK,V_MASK,W_MASK,P_MASK

   REAL(EB), POINTER, DIMENSION(:) :: WALL_WORK1,WALL_WORK2
   REAL(FB), POINTER, DIMENSION(:,:,:,:) :: QQ
   REAL(FB), POINTER, DIMENSION(:,:) :: PP,PPN
   INTEGER, POINTER, DIMENSION(:,:) :: IBK
   INTEGER, POINTER, DIMENSION(:,:,:) :: IBLK

   REAL(EB) :: DT,DT_PREV,DT_NEXT,DT_INIT
   REAL(EB) :: CFL,DIVMX,DIVMN,VN,RESMAX,PART_CFL
   INTEGER :: ICFL,JCFL,KCFL,IMX,JMX,KMX,IMN,JMN,KMN, I_VN,J_VN,K_VN,IRM,JRM,KRM

   INTEGER :: N_EDGES
   INTEGER, POINTER, DIMENSION(:,:) :: IJKE,EDGE_INDEX
   REAL(EB), POINTER, DIMENSION(:,:) :: TAU_E,OME_E
   INTEGER, POINTER, DIMENSION(:,:) :: EDGE_TYPE

   INTEGER :: MESH_LEVEL
   INTEGER :: IBAR,JBAR,KBAR,IBM1,JBM1,KBM1,IBP1,JBP1,KBP1
   INTEGER, POINTER, DIMENSION(:) :: RGB
   REAL(EB) :: DXI,DETA,DZETA,RDXI,RDETA,RDZETA, &
      DXMIN,DXMAX,DYMIN,DYMAX,DZMIN,DZMAX, &
      XS,XF,YS,YF,ZS,ZF,RDXINT,RDYINT,RDZINT,CELL_SIZE
   REAL(EB), POINTER, DIMENSION(:) :: R,RC,X,Y,Z,XC,YC,ZC,HX,HY,HZ, &
           DX,RDX,DXN,RDXN,DY,RDY,DYN,RDYN,DZ,RDZ,DZN,RDZN, &
```

```
                   CELLSI , CELLSJ , CELLSK,RRN
   REAL(FB) , POINTER, DIMENSION( : )  : : XPLT,YPLT,ZPLT

   INTEGER : : N_OBST
   TYPE(OBSTRUCTION_TYPE) , POINTER, DIMENSION( : )  : : OBSTRUCTION

   INTEGER : : N_VENT
   TYPE(VENTS_TYPE) , POINTER, DIMENSION( : )  : : VENTS

   INTEGER, POINTER, DIMENSION( : , : , : )  : : CELL_INDEX
   INTEGER, POINTER, DIMENSION( : )  : : I_CELL , J_CELL , K_CELL ,
        OBST_INDEX_C
   INTEGER, POINTER, DIMENSION( : , : )  : : WALL_INDEX
   LOGICAL, POINTER, DIMENSION( : )  : : SOLID ,EXTERIOR

   INTEGER : : N_INTERNAL_WALL_CELLS,N_EXTERNAL_WALL_CELLS,
        N_VIRTUAL_WALL_CELLS,N_GHOST_WALL_CELLS,  &
                    CELL_COUNT,WALL_COUNTER
   REAL(EB)  : : BC_CLOCK
   REAL(EB) , POINTER, DIMENSION( : , : )  : :
        EDGE_INTERPOLATION_FACTOR,AWM_AEROSOL
   REAL(EB) , POINTER, DIMENSION( : )      : : DUWDT,  &
                  D_CORR,DS_CORR,UVW_SAVE,U_GHOST,V_GHOST,W_GHOST
        TYPE(WALL_TYPE) , POINTER, DIMENSION( : )  : : WALL
   TYPE(OMESH_TYPE) , POINTER, DIMENSION( : )  : : OMESH
   TYPE(LAGRANGIAN_PARTICLE_TYPE) , POINTER, DIMENSION( : )  : :
        LAGRANGIAN_PARTICLE
   INTEGER : : NLP,NLPDIM
   TYPE(HUMAN_TYPE) , POINTER, DIMENSION( : )  : : HUMAN
   INTEGER : : N_HUMANS,N_HUMANS_DIM
   TYPE(HUMAN_GRID_TYPE) , POINTER, DIMENSION( : , : )  : : HUMAN_GRID

   INTEGER : : N_SLCF
   TYPE(SLICE_TYPE) , POINTER, DIMENSION( : )  : : SLICE

   INTEGER, POINTER, DIMENSION( : , : )  : : INC
   INTEGER : : NPATCH

   REAL(EB) , POINTER, DIMENSION( : , : , : , : )  : : UIID
   INTEGER : : RAD_CALL_COUNTER,ANGLE_INC_COUNTER

   INTEGER, POINTER, DIMENSION( : , : , : )  : : INTERPOLATED_MESH

```

```fortran
    CHARACTER(80), POINTER, DIMENSION(:) :: STRING
    INTEGER :: N_STRINGS,N_STRINGS_MAX

!rm ->
!   REAL(EB), POINTER, DIMENSION(:,:,:,:) :: DMPVDT_FM_VEG
    INTEGER, POINTER, DIMENSION(:,:,:) :: K_AGL_SLICE
    REAL(EB),POINTER, DIMENSION(:,:) :: LS_Z_TERRAIN,VEG_DRAG
    INTEGER :: N_TERRAIN_SLCF
    REAL(EB) :: VEG_CLOCK_BC !surf veg
!rm <-

END TYPE MESH_TYPE

TYPE (MESH_TYPE), SAVE, DIMENSION(:), ALLOCATABLE, TARGET ::
    MESHES

END MODULE MESH_VARIABLES


MODULE MESH_POINTERS

USE PRECISION_PARAMETERS
USE MESH_VARIABLES
IMPLICIT NONE

REAL(EB), POINTER, DIMENSION(:,:,:) :: &
                U,V,W,US,VS,WS,DDDT,D,DS,H,HS,KRES,FVX,FVY,FVZ,RHO,
                    RHOS, &
                MU,TMP,Q,FRHO,KAPPA,QR,QR_W,UII,RSUM,D_LAGRANGIAN,
                    D_REACTION, &
                CSD2,MTR,MSR,WEM,MIX_TIME,STRAIN_RATE,KFST4,
                    RHO_H_S_OVER_PBAR,D_RHSOP_DT,D_RHSOP_DT_S,NU_EDDY
                    !ADDED
REAL(EB), POINTER, DIMENSION(:,:,:,:) :: ZZ,ZZS,DEL_RHO_D_DEL_Z
REAL(EB), POINTER, DIMENSION(:,:,:,:) :: AVG_DROP_DEN,
    AVG_DROP_TMP,AVG_DROP_RAD,AVG_DROP_AREA
REAL(EB), POINTER, DIMENSION(:,:,:)   :: AVG_DROP_DEN_ALL
REAL(EB), POINTER, DIMENSION(:,:) :: UVW_GHOST
REAL(EB), POINTER :: POIS_PTB,POIS_ERR
REAL(EB), POINTER, DIMENSION(:) :: SAVE1,SAVE2,WORK
REAL(EB), POINTER, DIMENSION(:,:,:) :: PRHS
REAL(EB), POINTER, DIMENSION(:,:) :: BXS,BXF,BYS,BYF,BZS,BZF,
    BXST,BXFT,BYST,BYFT,BZST,BZFT
```

```
INTEGER, POINTER :: LSAVE,LWORK,LBC,MBC,NBC,ITRN,JTRN,KTRN,IPS
REAL(EB), POINTER, DIMENSION(:) :: P_0,RHO_0,TMP_0,D_PBAR_DT,
    D_PBAR_S_DT,U_LEAK,U_DUCT
REAL(EB), POINTER, DIMENSION(:,:) :: PBAR,PBAR_S,R_PBAR
INTEGER, POINTER, DIMENSION(:,:,:) :: PRESSURE_ZONE
INTEGER, POINTER, DIMENSION(:) :: PRESSURE_BC_INDEX
REAL(EB), POINTER, DIMENSION(:,:,:) :: WORK1,WORK2,WORK3,WORK4,
    WORK5,WORK6,WORK7,WORK8

REAL(EB), POINTER, DIMENSION(:,:,:) :: TURB_WORK1,TURB_WORK2,
    TURB_WORK3,TURB_WORK4
REAL(EB), POINTER, DIMENSION(:,:,:) :: TURB_WORK5,TURB_WORK6,
    TURB_WORK7,TURB_WORK8
REAL(EB), POINTER, DIMENSION(:,:,:) :: TURB_WORK9,TURB_WORK10
REAL(EB), POINTER, DIMENSION(:) :: TURB_WORK11,TURB_WORK12

REAL(EB), POINTER, DIMENSION(:,:,:) :: IBM_SAVE1,IBM_SAVE2,
    IBM_SAVE3,IBM_SAVE4,IBM_SAVE5,IBM_SAVE6
INTEGER, POINTER, DIMENSION(:,:,:) :: U_MASK,V_MASK,W_MASK,
    P_MASK

REAL(EB), POINTER, DIMENSION(:) :: WALL_WORK1,WALL_WORK2
REAL(FB), POINTER, DIMENSION(:,:,:,:) :: QQ
REAL(FB), POINTER, DIMENSION(:,:) :: PP,PPN
INTEGER, POINTER, DIMENSION(:,:) :: IBK
INTEGER, POINTER, DIMENSION(:,:,:) :: IBLK
REAL(EB), POINTER :: DT,DT_PREV,DT_NEXT,DT_INIT
REAL(EB), POINTER :: CFL,DIVMX,DIVMN,VN,RESMAX,PART_CFL
INTEGER, POINTER :: ICFL,JCFL,KCFL,IMX,JMX,KMX,IMN,JMN,KMN,I_VN
    ,J_VN,K_VN,IRM,JRM,KRM
INTEGER, POINTER :: N_EDGES
INTEGER, POINTER, DIMENSION(:,:) :: IJKE,EDGE_INDEX
REAL(EB), POINTER, DIMENSION(:,:) :: TAU_E,OME_E
INTEGER, POINTER, DIMENSION(:,:) :: EDGE_TYPE

INTEGER, POINTER :: MESH_LEVEL
INTEGER, POINTER :: IBAR,JBAR,KBAR,IBM1,JBM1,KBM1,IBP1,JBP1,
    KBP1
INTEGER, POINTER, DIMENSION(:) :: RGB
REAL(EB), POINTER :: DXI,DETA,DZETA,RDXI,RDETA,RDZETA, &
    DXMIN,DXMAX,DYMIN,DYMAX,DZMIN,DZMAX, &
    XS,XF,YS,YF,ZS,ZF,RDXINT,RDYINT,RDZINT,CELL_SIZE
```

```
REAL(EB) , POINTER, DIMENSION(:) :: R,RC,X,Y,Z,XC,YC,ZC,HX,HY,HZ
     , &
               DX,RDX,DXN,RDXN,DY,RDY,DYN,RDYN,DZ,RDZ,DZN,RDZN, &
               CELLSI,CELLSJ,CELLSK,RRN
REAL(FB) , POINTER, DIMENSION(:) :: XPLT,YPLT,ZPLT
INTEGER, POINTER :: N_OBST
TYPE(OBSTRUCTION_TYPE) , POINTER, DIMENSION(:) :: OBSTRUCTION
INTEGER, POINTER :: N_VENT
TYPE(VENTS_TYPE) , POINTER, DIMENSION(:) :: VENTS
INTEGER, POINTER, DIMENSION(:,:,:) :: CELL_INDEX
INTEGER, POINTER, DIMENSION(:) :: I_CELL,J_CELL,K_CELL,
     OBST_INDEX_C
INTEGER, POINTER, DIMENSION(:,:) :: WALL_INDEX
LOGICAL, POINTER, DIMENSION(:) :: SOLID,EXTERIOR
INTEGER, POINTER :: N_INTERNAL_WALL_CELLS,N_EXTERNAL_WALL_CELLS
     ,N_VIRTUAL_WALL_CELLS,N_GHOST_WALL_CELLS, &
                         CELL_COUNT,WALL_COUNTER
REAL(EB) ,POINTER :: BC_CLOCK
REAL(EB) , POINTER, DIMENSION(:,:) :: EDGE_INTERPOLATION_FACTOR,
     AWM_AEROSOL
REAL(EB) , POINTER, DIMENSION(:)     :: DUWDT, &
               D_CORR,DS_CORR,UVW_SAVE,U_GHOST,V_GHOST,W_GHOST
TYPE(WALL_TYPE) , POINTER, DIMENSION(:) :: WALL
TYPE(OMESH_TYPE) , POINTER, DIMENSION(:) :: OMESH
TYPE(LAGRANGIAN_PARTICLE_TYPE) , POINTER, DIMENSION(:) ::
     LAGRANGIAN_PARTICLE
INTEGER, POINTER :: NLP,NLPDIM
TYPE(HUMAN_TYPE) , POINTER, DIMENSION(:) :: HUMAN
INTEGER, POINTER :: N_HUMANS,N_HUMANS_DIM
TYPE(HUMAN_GRID_TYPE) , POINTER, DIMENSION(:,:) :: HUMAN_GRID
INTEGER, POINTER :: N_SLCF
TYPE(SLICE_TYPE) , POINTER, DIMENSION(:) :: SLICE
INTEGER, POINTER, DIMENSION(:,:) :: INC
INTEGER, POINTER :: NPATCH
REAL(EB) , POINTER, DIMENSION(:,:,:,:) :: UIID
INTEGER,  POINTER :: RAD_CALL_COUNTER,ANGLE_INC_COUNTER
INTEGER, POINTER, DIMENSION(:,:,:) :: INTERPOLATED_MESH
CHARACTER(80) , POINTER, DIMENSION(:) :: STRING
INTEGER, POINTER :: N_STRINGS,N_STRINGS_MAX
!rm ->
!REAL(EB) , POINTER, DIMENSION(:,:,:,:) :: DMPVDT_FM_VEG
INTEGER, POINTER, DIMENSION(:,:,:) :: K_AGL_SLICE
REAL(EB) , POINTER,DIMENSION(:,:) :: LS_Z_TERRAIN,VEG_DRAG
```

```
INTEGER, POINTER :: N_TERRAIN_SLCF
REAL(EB) , POINTER :: VEG_CLOCK_BC   !surf veg
!rm <-

CONTAINS
```

B.4 init.f90

Modified subroutine in init.f90:

```
SUBROUTINE INITIALIZE_MESH_VARIABLES(NM)

USE PHYSICAL_FUNCTIONS, ONLY: GET_VISCOSITY,
    GET_SPECIFIC_GAS_CONSTANT,GET_SPECIFIC_HEAT
USE GEOMETRY_FUNCTIONS, ONLY: ASSIGN_PRESSURE_ZONE
USE RADCONS, ONLY: UIIDIM
USE CONTROL_VARIABLES
INTEGER :: N,I,J,K,II,JJ,KK,IPTS,JPTS,KPTS,N_EDGES_DIM,
    N_TOTAL_WALL_CELLS,IW,IWE,IWG,IC,SURF_INDEX,IOR,IOPZ, &
               IERR,IB,JB,KB,IPZ
INTEGER, INTENT(IN) :: NM
REAL(EB) :: MU_N,ZZ_GET(0:N_TRACKED_SPECIES),VC,RTRM,CP,CS,
    DELTA
INTEGER, POINTER :: IBP1, JBP1, KBP1,IBAR, JBAR, KBAR, N_EDGES
REAL(EB),POINTER :: XS,XF,YS,YF,ZS,ZF
TYPE (INITIALIZATION_TYPE), POINTER :: IN
TYPE (P_ZONE_TYPE), POINTER :: PZ
TYPE (DEVICE_TYPE), POINTER :: DV
TYPE (VENTS_TYPE), POINTER :: VT

IERR = 0
M => MESHES(NM)
IBP1 =>M%IBP1
JBP1 =>M%JBP1
KBP1 =>M%KBP1
IBAR =>M%IBAR
JBAR =>M%JBAR
KBAR =>M%KBAR
N_EDGES=>M%N_EDGES
XS=>M%XS
YS=>M%YS
ZS=>M%ZS
XF=>M%XF
YF=>M%YF
ZF=>M%ZF
ALLOCATE(M%RHO(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','RHO',IZERO)
ALLOCATE(M%RHOS(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','RHOS',IZERO)
M%RHOS = RHOA
```

```
ALLOCATE(M%TMP(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','TMP',IZERO)
ALLOCATE(M%FRHO(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','FRHO',IZERO)
ALLOCATE(M%U(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','U',IZERO)
ALLOCATE(M%V(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','V',IZERO)
ALLOCATE(M%W(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','W',IZERO)
ALLOCATE(M%US(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','US',IZERO)
ALLOCATE(M%VS(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','VS',IZERO)
ALLOCATE(M%WS(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','WS',IZERO)
ALLOCATE(M%FVX(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','FVX',IZERO)
ALLOCATE(M%FVY(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','FVY',IZERO)
ALLOCATE(M%FVZ(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','FVZ',IZERO)
ALLOCATE(M%H(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','H',IZERO)
ALLOCATE(M%HS(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','HS',IZERO)
ALLOCATE(M%KRES(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','KRES',IZERO)
ALLOCATE(M%DDDT(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','DDDT',IZERO)
ALLOCATE(M%D(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','D',IZERO)
ALLOCATE(M%DS(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','DS',IZERO)
ALLOCATE(M%MU(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','MU',IZERO)
ALLOCATE(M%STRAIN_RATE(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','STRAIN_RATE',IZERO)
ALLOCATE(M%NU_EDDY(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO) !ADDED
CALL ChkMemErr('INIT','NU_EDDY',IZERO) !ADDED
ALLOCATE(M%CSD2(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','CS',IZERO)

```

```
IF (.NOT.EVACUATION_ONLY(NM)) THEN
   ALLOCATE(M%Q(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
   CALL ChkMemErr('INIT','Q',IZERO)
ENDIF

ALLOCATE(M%MIX_TIME(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','MIX_TIME',IZERO)
M%MIX_TIME=M%DT

! Background pressure, temperature, density as a function of
    height (Z coordinate)

ALLOCATE(   M%PBAR(0:KBP1,0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('INIT','PBAR',IZERO)
ALLOCATE(   M%PBAR_S(0:KBP1,0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('INIT','PBAR_S',IZERO)
ALLOCATE(   M%R_PBAR(0:KBP1,0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('INIT','R_PBAR',IZERO)
ALLOCATE(   M%D_PBAR_DT(0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('INIT','D_PBAR_DT',IZERO)
ALLOCATE(   M%D_PBAR_S_DT(0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('INIT','D_PBAR_S_DT',IZERO)
ALLOCATE(M%P_0(0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','P_0',IZERO)
ALLOCATE(M%TMP_0(0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','TMP_0',IZERO)
ALLOCATE(M%RHO_0(0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','RHO_0',IZERO)

! Leaks

ALLOCATE(   M%U_LEAK(0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('INIT','U_LEAK',IZERO)
M%U_LEAK = 0._EB

! Allocate species arrays

IF (N_TRACKED_SPECIES>0 .AND. .NOT.EVACUATION_ONLY(NM)) THEN
   ALLOCATE( M%ZZ(0:IBP1,0:JBP1,0:KBP1,N_TRACKED_SPECIES),STAT=
      IZERO)
   CALL ChkMemErr('INIT','ZZ',IZERO)
   M%ZZ = 0._EB
```

```
    ALLOCATE(M%ZZS(0:IBP1,0:JBP1,0:KBP1,N_TRACKED_SPECIES),STAT=
        IZERO)
    CALL ChkMemErr('INIT','ZZS',IZERO)
    M%ZZS = 0._EB
    ALLOCATE(M%DEL_RHO_D_DEL_Z(0:IBP1,0:JBP1,0:KBP1,
        N_TRACKED_SPECIES),STAT=IZERO)
    CALL ChkMemErr('INIT','DEL_RHO_D_DEL_Z',IZERO)
    M%DEL_RHO_D_DEL_Z = 0._EB
ENDIF

ALLOCATE(M%RSUM(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','RSUM',IZERO)
M%RSUM = RSUM0

! Allocate reaction divergence

IF (N_REACTIONS > 0) THEN
    ALLOCATE(M%D_REACTION(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','D_REACTION',IZERO)
    M%D_REACTION = 0._EB
ENDIF

! Enthalpy arrays (experimental)

IF (ENTHALPY_TRANSPORT) THEN
    ALLOCATE(M%RHO_H_S_OVER_PBAR(0:IBP1,0:JBP1,0:KBP1),STAT=
        IZERO)
    CALL ChkMemErr('INIT','RHO_H_S_OVER_PBAR',IZERO)
    M%RHO_H_S_OVER_PBAR = 0._EB ! initialized in DENSITY
    ALLOCATE(M%D_RHSOP_DT(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','D_RHSOP_DT',IZERO)
    M%D_RHSOP_DT = 0._EB
    ALLOCATE(M%D_RHSOP_DT_S(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','D_RHSOP_DT_S',IZERO)
    M%D_RHSOP_DT_S = 0._EB
ENDIF

! Allocate water mass arrays if sprinklers are present

IF (PARTICLE_FILE) PARTICLE_TAG = NM

IF (N_LAGRANGIAN_CLASSES>0 .AND. .NOT. EVACUATION_ONLY(NM))
    THEN
```

```
   ALLOCATE(M%AVG_DROP_DEN_ALL(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO
      )
   CALL ChkMemErr('INIT','AVG_DROP_DEN_ALL',IZERO)
   M%AVG_DROP_DEN_ALL=0._EB
ENDIF

IF (N_LP_ARRAY_INDICES>0 .AND. .NOT.EVACUATION_ONLY(NM)) THEN
   ALLOCATE(M%QR_W(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
   CALL ChkMemErr('INIT','QR_W',IZERO)
   M%QR_W = 0._EB
   ALLOCATE(M%AVG_DROP_DEN(0:IBP1,0:JBP1,0:KBP1,
      N_LP_ARRAY_INDICES),STAT=IZERO)
   CALL ChkMemErr('INIT','AVG_DROP_DEN',IZERO)
   M%AVG_DROP_DEN=0._EB
   ALLOCATE(M%AVG_DROP_AREA(0:IBP1,0:JBP1,0:KBP1,
      N_LP_ARRAY_INDICES),STAT=IZERO)
   CALL ChkMemErr('INIT','AVG_DROP_AREA',IZERO)
   M%AVG_DROP_AREA=0._EB
   ALLOCATE(M%AVG_DROP_TMP(0:IBP1,0:JBP1,0:KBP1,
      N_LP_ARRAY_INDICES),STAT=IZERO)
   CALL ChkMemErr('INIT','AVG_DROP_TMP',IZERO)
   M%AVG_DROP_TMP=TMPM
   ALLOCATE(M%AVG_DROP_RAD(0:IBP1,0:JBP1,0:KBP1,
      N_LP_ARRAY_INDICES),STAT=IZERO)
   CALL ChkMemErr('INIT','AVG_DROP_RAD',IZERO)
   M%AVG_DROP_RAD=0._EB
   ALLOCATE(M%D_LAGRANGIAN(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
   CALL ChkMemErr('INIT','D_LAGRANGIAN',IZERO)
   M%D_LAGRANGIAN = 0._EB
ENDIF

! If radiation absorption desired allocate arrays

IF (.NOT.EVACUATION_ONLY(NM)) THEN
   ALLOCATE(M%QR(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
   CALL ChkMemErr('INIT','QR',IZERO)
   M%QR = 0._EB
   ALLOCATE(M%KAPPA(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
   CALL ChkMemErr('INIT','KAPPA',IZERO)
   M%KAPPA = KAPPA0
   ALLOCATE(M%UII(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
   CALL ChkMemErr('INIT','UII',IZERO)
   M%UII = 0._EB
```

```
    ALLOCATE(M%KFST4(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','KFST4',IZERO)
    M%KFST4 = 0._EB
ELSE
    ALLOCATE(M%QR(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','QR',IZERO)
    M%QR = 0._EB
ENDIF

! Work arrays

ALLOCATE(M%WORK1(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','WORK1',IZERO)
ALLOCATE(M%WORK2(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','WORK2',IZERO)
ALLOCATE(M%WORK3(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','WORK3',IZERO)
ALLOCATE(M%WORK4(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','WORK4',IZERO)
ALLOCATE(M%WORK5(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','WORK5',IZERO)
ALLOCATE(M%WORK6(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','WORK6',IZERO)
ALLOCATE(M%WORK7(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','WORK7',IZERO)
ALLOCATE(M%WORK8(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','WORK8',IZERO)

IF (IMMERSED_BOUNDARY_METHOD==2) THEN
    ALLOCATE(M%IBM_SAVE1(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','IBM_SAVE1',IZERO)
    ALLOCATE(M%IBM_SAVE2(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','IBM_SAVE2',IZERO)
    ALLOCATE(M%IBM_SAVE3(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','IBM_SAVE3',IZERO)
    ALLOCATE(M%IBM_SAVE4(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','IBM_SAVE4',IZERO)
    ALLOCATE(M%IBM_SAVE5(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','IBM_SAVE5',IZERO)
    ALLOCATE(M%IBM_SAVE6(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT','IBM_SAVE6',IZERO)
ENDIF

```

```
! Boundary file patch counter

ALLOCATE(M%INC(-3:3,0:M%N_OBST),STAT=IZERO)
CALL ChkMemErr('INIT','INC',IZERO)

! Initialize background pressure, temperature and density

M%D_PBAR_DT    = 0._EB
M%D_PBAR_S_DT = 0._EB

IF (STRATIFICATION .AND. .NOT.EVACUATION_ONLY(NM)) THEN
   DO K=0,M%KBP1
      M%TMP_0(K) = TMPA + LAPSE_RATE*(M%ZC(K)-GROUND_LEVEL)
      IF (ABS(LAPSE_RATE)>ZERO_P) THEN
         M%P_0(K) = P_INF*(M%TMP_0(K)/M%TMP_0(0))**(GVEC(3)/
            RSUM0/LAPSE_RATE)
      ELSE
         M%P_0(K) = P_INF*EXP(GVEC(3)*(M%ZC(K)-GROUND_LEVEL)/(
            RSUM0*TMPA))
      ENDIF
   ENDDO
ELSE
   M%TMP_0(:) = TMPA
   M%P_0(:)   = P_INF
ENDIF
DO K=0,M%KBP1
   M%PBAR(K,:)   = M%P_0(K)
   M%PBAR_S(K,:) = M%P_0(K)
   M%RHO_0(K)    = M%P_0(K)/(M%TMP_0(K)*RSUM0)
ENDDO

! Initialize various time step variables

M%DT_PREV = M%DT
M%DT_NEXT = M%DT
M%DT_INIT = M%DT

! Initialize major arrays

DO K=0,M%KBP1
   M%RHO(:,:,K) = M%RHO_0(K)
   M%TMP(:,:,K) = M%TMP_0(K)
ENDDO
```

```
IF (.NOT.EVACUATION_ONLY(NM)) M%FRHO      = 0._EB
M%U         = U0
M%V         = V0
M%W         = W0
M%US        = U0
M%VS        = V0
M%WS        = W0
M%FVX    = 0._EB
M%FVY    = 0._EB
M%FVZ    = 0._EB
M%H      = H0
M%HS     = H0
M%KRES   = 0._EB

M%DDDT   = 0._EB
M%D      = 0._EB
M%DS     = 0._EB
IF (.NOT.EVACUATION_ONLY(NM)) THEN
   M%Q       = 0._EB
ENDIF
IF (EVACUATION_ONLY(NM)) THEN
   M%U         = 0._EB
   M%V         = 0._EB
   M%W         = 0._EB
   M%US        = 0._EB
   M%VS        = 0._EB
   M%WS        = 0._EB
   M%H         = 0._EB
   M%HS        = 0._EB
ENDIF
IF (N_TRACKED_SPECIES > 0 .AND. .NOT.EVACUATION_ONLY(NM)) M%
   DEL_RHO_D_DEL_Z = 0._EB

! Viscosity

IF (N_TRACKED_SPECIES>0) ZZ_GET(1:N_TRACKED_SPECIES) =
   SPECIES_MIXTURE(1:N_TRACKED_SPECIES)%ZZ0
CALL GET_VISCOSITY(ZZ_GET,MU_N,TMPA)
M%MU = MU_N

CS = C_SMAGORINSKY
IF (EVACUATION_ONLY(NM)) CS=0.9_EB
DO K=0,KBP1
```

```
   DO J=0,JBP1
      DO I=0,IBP1
         IF (TWO_D) THEN
            DELTA = MAX(M%DX(I),M%DZ(K))
         ELSE
            DELTA = MAX(M%DX(I),M%DY(J),M%DZ(K))
         ENDIF
         M%CSD2(I,J,K) = (CS*DELTA)**2
      ENDDO
   ENDDO
ENDDO

! Initialize mass fraction arrays

IF (N_TRACKED_SPECIES > 0 .AND. .NOT.EVACUATION_ONLY(NM)) THEN
   DO N=1,N_TRACKED_SPECIES
      M%ZZ(:,:,:,N)  = SPECIES_MIXTURE(N)%ZZ0
      M%ZZS(:,:,:,N) = SPECIES_MIXTURE(N)%ZZ0
   ENDDO
ENDIF

! Initialize pressure ZONEs

ALLOCATE(M%PRESSURE_ZONE(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','PRESSURE_ZONE',IZERO)
M%PRESSURE_ZONE = 0
ZONE_LOOP: DO N=1,N_ZONE
   IF (EVACUATION_ONLY(NM)) CYCLE ZONE_LOOP
   PZ => P_ZONE(N)
   DO K=0,KBP1
      DO J=0,JBP1
         DO I=0,IBP1
            IF (M%PRESSURE_ZONE(I,J,K)==N) CYCLE
            IF (M%XC(I) > PZ%X1 .AND. M%XC(I) < PZ%X2 .AND. &
                M%YC(J) > PZ%Y1 .AND. M%YC(J) < PZ%Y2 .AND. &
                M%ZC(K) > PZ%Z1 .AND. M%ZC(K) < PZ%Z2) THEN
                M%PRESSURE_ZONE(I,J,K) = N
                DO IOPZ=0,N_ZONE
                   IF (PZ%LEAK_AREA(IOPZ) > 0._EB)
                      ACTUAL_LEAK_AREA(N,IOPZ) = PZ%LEAK_AREA(
                      IOPZ)
                   IF (PZ%LEAK_AREA(IOPZ) > 0._EB)
                      ACTUAL_LEAK_AREA(IOPZ,N) = PZ%LEAK_AREA(
                           IOPZ)
                ENDDO
                IF (.NOT.M%SOLID(M%CELL_INDEX(I,J,K))) CALL
                   ASSIGN_PRESSURE_ZONE(NM,M%XC(I),M%YC(J),M%ZC
                   (K),N)
            ENDIF
         ENDDO
      ENDDO
   ENDDO
ENDDO ZONE_LOOP


! Over-ride default ambient conditions with user-prescribed
   INITializations

DO N=1,N_INIT
   IF (EVACUATION_ONLY(NM)) CYCLE
   IN => INITIALIZATION(N)
   DO K=0,KBP1
      DO J=0,JBP1
         DO I=0,IBP1
            IF (M%XC(I) > IN%X1 .AND. M%XC(I) < IN%X2 .AND. &
                M%YC(J) > IN%Y1 .AND. M%YC(J) < IN%Y2 .AND. &
                M%ZC(K) > IN%Z1 .AND. M%ZC(K) < IN%Z2) THEN
               M%TMP(I,J,K)            = IN%TEMPERATURE
               M%RHO(I,J,K)            = IN%DENSITY
               IF (N_TRACKED_SPECIES>0) M%ZZ(I,J,K,1:
                  N_TRACKED_SPECIES) = IN%MASS_FRACTION(1:
                  N_TRACKED_SPECIES)
               IF (IN%ADJUST_DENSITY)     M%RHO(I,J,K) = M%RHO(
                  I,J,K)*M%P_0(K)/P_INF
               IF (IN%ADJUST_TEMPERATURE) M%TMP(I,J,K) = M%TMP(
                  I,J,K)*M%P_0(K)/P_INF
               M%Q(I,J,K) = IN%HRRPUV
            ENDIF
         ENDDO
      ENDDO
   ENDDO
ENDDO

! Compute molecular weight term RSUM=R0*SUM(Y_i/M_i)

IF (N_TRACKED_SPECIES>0 .AND. .NOT.EVACUATION_ONLY(NM)) THEN
```

```
    DO K=1,KBAR
        DO J=1,JBAR
            DO I=1,IBAR
                ZZ_GET(1:N_TRACKED_SPECIES) = M%ZZ(I,J,K,1:N_TRACKED_SPECIES)
                CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET,M%RSUM(I,J,K))
            ENDDO
        ENDDO
    ENDDO
ENDIF

! Allocate and Initialize Mesh-Dependent Radiation Arrays

M%QR     = 0._EB
IF (.NOT.EVACUATION_ONLY(NM)) THEN
    M%KAPPA = KAPPA0
    M%UII    = 4._EB*SIGMA*TMPA4
ENDIF
M%ANGLE_INC_COUNTER = 0
M%RAD_CALL_COUNTER  = 0
IF (RADIATION .AND. .NOT.EVACUATION_ONLY(NM)) THEN
    ALLOCATE(M%UIID(0:M%IBP1,0:M%JBP1,0:M%KBP1,1:UIIDIM),STAT=IZERO)
    CALL ChkMemErr('INIT','UIID',IZERO)
    M%UIID = 0.
ENDIF

! General work arrays

M%WORK1 = 0._EB
M%WORK2 = 0._EB
M%WORK3 = 0._EB
M%WORK4 = 0._EB
M%WORK5 = 0._EB
M%WORK6 = 0._EB
IF (.NOT.EVACUATION_ONLY(NM)) M%WORK7 = 0._EB

! Immersed Boundary Method

IF (IMMERSED_BOUNDARY_METHOD==2) THEN
    M%IBM_SAVE1 = 0._EB
    M%IBM_SAVE2 = 0._EB
    M%IBM_SAVE3 = 0._EB
    M%IBM_SAVE4 = 0._EB
    M%IBM_SAVE5 = 0._EB
    M%IBM_SAVE6 = 0._EB
ENDIF

IF (IMMERSED_BOUNDARY_METHOD>=0) THEN
    ALLOCATE(M%U_MASK(0:M%IBP1,0:M%JBP1,0:M%KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT_IBM','U_MASK',IZERO)
    ALLOCATE(M%V_MASK(0:M%IBP1,0:M%JBP1,0:M%KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT_IBM','V_MASK',IZERO)
    ALLOCATE(M%W_MASK(0:M%IBP1,0:M%JBP1,0:M%KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT_IBM','W_MASK',IZERO)
    ALLOCATE(M%P_MASK(0:M%IBP1,0:M%JBP1,0:M%KBP1),STAT=IZERO)
    CALL ChkMemErr('INIT_IBM','P_MASK',IZERO)
    M%U_MASK=1
    M%V_MASK=1
    M%W_MASK=1
    M%P_MASK=1
ENDIF

! Determine the total number of wall cells to allocate

M%N_INTERNAL_WALL_CELLS = 0

OBST_LOOP_1: DO N=1,M%N_OBST
    OB=>M%OBSTRUCTION(N)
    IF (OB%CONSUMABLE .AND. .NOT.EVACUATION_ONLY(NM)) THEN
        IB    = OB%I2-OB%I1
        JB    = OB%J2-OB%J1
        KB    = OB%K2-OB%K1
        M%N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS + 2*(
            MAX(1,IB)*JB*KB + MAX(1,JB)*IB*KB + MAX(1,KB)*IB*JB)
    ELSE
        DO K=OB%K1+1,OB%K2
            DO J=OB%J1+1,OB%J2
                IC = M%CELL_INDEX(OB%I1   ,J,K)
                IF (.NOT.M%SOLID(IC).OR.M%OBSTRUCTION(M%
                    OBST_INDEX_C(IC))%REMOVABLE) M%
                    N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS
                    +1
                IC = M%CELL_INDEX(OB%I2+1,J,K)
```

```
               IF (.NOT.M%SOLID(IC).OR.M%OBSTRUCTION(M%
                  OBST_INDEX_C(IC))%REMOVABLE) M%
                  N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS
                  +1
            ENDDO
         ENDDO
         DO K=OB%K1+1,OB%K2
            DO I=OB%I1+1,OB%I2
               IC = M%CELL_INDEX(I,OB%J1   ,K)
               IF (.NOT.M%SOLID(IC).OR.M%OBSTRUCTION(M%
                  OBST_INDEX_C(IC))%REMOVABLE) M%
                  N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS
                  +1
               IC = M%CELL_INDEX(I,OB%J2+1,K)
               IF (.NOT.M%SOLID(IC).OR.M%OBSTRUCTION(M%
                  OBST_INDEX_C(IC))%REMOVABLE) M%
                  N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS
                  +1
            ENDDO
         ENDDO
         DO J=OB%J1+1,OB%J2
            DO I=OB%I1+1,OB%I2
               IC = M%CELL_INDEX(I,J,OB%K1   )
               IF (.NOT.M%SOLID(IC).OR.M%OBSTRUCTION(M%
                  OBST_INDEX_C(IC))%REMOVABLE) M%
                  N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS
                  +1
               IC = M%CELL_INDEX(I,J,OB%K2+1)
               IF (.NOT.M%SOLID(IC).OR.M%OBSTRUCTION(M%
                  OBST_INDEX_C(IC))%REMOVABLE) M%
                  N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS
                  +1
            ENDDO
         ENDDO
      ENDIF
ENDDO OBST_LOOP_1

! Add wall cells for VIRTUAL devices

M%N_VIRTUAL_WALL_CELLS = 0

DO N=1,N_DEVC
   DV => DEVICE(N)
   IF (DV%MESH/=NM) CYCLE
   IF (EVACUATION_ONLY(NM)) CYCLE
   IF (DV%QUANTITY/='CABLE_TEMPERATURE') CYCLE
   M%N_VIRTUAL_WALL_CELLS = M%N_VIRTUAL_WALL_CELLS + 1
ENDDO

! Compute the number of ghost wall cells (external wall cells
    outside the computational domain)

M%N_GHOST_WALL_CELLS = 8*(IBP1+JBP1+KBP1)

! Allocate arrays indexed by wall cells (IW). Note the order of
    the cells in the overall array.

N_TOTAL_WALL_CELLS = M%N_EXTERNAL_WALL_CELLS + M%
   N_INTERNAL_WALL_CELLS + M%N_VIRTUAL_WALL_CELLS + M%
   N_GHOST_WALL_CELLS

ALLOCATE(M%WALL(0:N_TOTAL_WALL_CELLS),STAT=IZERO)
CALL ChkMemErr('INIT','WALL',IZERO)
M%WALL%RHO_F = RHOA
M%WALL%ONE_D%EMISSIVITY = 1._EB
M%WALL%U_TAU = 0._EB

NOT_EVAC_IF_1: IF (.NOT.EVACUATION_ONLY(NM)) THEN

   M%WALL%TMP_F = TMPA
   M%WALL%TMP_B = TMPA
   DO IW=1,N_TOTAL_WALL_CELLS
      ALLOCATE(M%WALL(IW)%ZZ_F(N_TRACKED_SPECIES),STAT=IZERO)
      CALL ChkMemErr('INIT','WALL(IW)%ZZ_F',IZERO)
      M%WALL(IW)%ZZ_F(1:N_TRACKED_SPECIES) = SPECIES_MIXTURE(1:
         N_TRACKED_SPECIES)%ZZ0
   ENDDO
   M%WALL%ONE_D%QRADIN = SIGMA*TMPA4
   M%WALL%ONE_D%QRADOUT = SIGMA*TMPA4
   M%WALL%ONE_D%QCONF = 0._EB
   M%WALL%ONE_D%HEAT_TRANS_COEF = 0._EB

   ALLOCATE(M%D_CORR(M%N_EXTERNAL_WALL_CELLS),STAT=IZERO)
   CALL ChkMemErr('INIT','D_CORR',IZERO)
   ALLOCATE(M%DS_CORR(M%N_EXTERNAL_WALL_CELLS),STAT=IZERO)
   CALL ChkMemErr('INIT','DS_CORR',IZERO)
```

```
    M%D_CORR = 0._EB
    M%DS_CORR = 0._EB
    ALLOCATE(M%UVW_SAVE(M%N_EXTERNAL_WALL_CELLS),STAT=IZERO)
    CALL ChkMemErr('INIT','UVW_SAVE',IZERO)
    M%UVW_SAVE = 0._EB

    ALLOCATE(M%U_GHOST(M%N_EXTERNAL_WALL_CELLS),STAT=IZERO)
    CALL ChkMemErr('INIT','U_GHOST',IZERO)
    ALLOCATE(M%V_GHOST(M%N_EXTERNAL_WALL_CELLS),STAT=IZERO)
    CALL ChkMemErr('INIT','V_GHOST',IZERO)
    ALLOCATE(M%W_GHOST(M%N_EXTERNAL_WALL_CELLS),STAT=IZERO)
    CALL ChkMemErr('INIT','W_GHOST',IZERO)
    M%U_GHOST = 0._EB
    M%V_GHOST = 0._EB
    M%W_GHOST = 0._EB

ENDIF NOT_EVAC_IF_1

! Allocate arrays for turbulent inflow boundary conditions (
    experimental)

VENT_LOOP: DO N=1,M%N_VENT
    VT => M%VENTS(N)
    EDDY_IF: IF (VT%N_EDDY>0) THEN
        SELECT CASE(ABS(VT%IOR))
            CASE(1)
                ALLOCATE(VT%U_EDDY(VT%J1+1:VT%J2,VT%K1+1:VT%K2),
                    STAT=IZERO)
                CALL ChkMemErr('READ_VENT','U_EDDY',IZERO)
                ALLOCATE(VT%V_EDDY(VT%J1+1:VT%J2,VT%K1+1:VT%K2),
                    STAT=IZERO)
                CALL ChkMemErr('READ_VENT','V_EDDY',IZERO)
                ALLOCATE(VT%W_EDDY(VT%J1+1:VT%J2,VT%K1+1:VT%K2),
                    STAT=IZERO)
                CALL ChkMemErr('READ_VENT','W_EDDY',IZERO)
            CASE(2)
                ALLOCATE(VT%U_EDDY(VT%I1+1:VT%I2,VT%K1+1:VT%K2),
                    STAT=IZERO)
                CALL ChkMemErr('READ_VENT','U_EDDY',IZERO)
                ALLOCATE(VT%V_EDDY(VT%I1+1:VT%I2,VT%K1+1:VT%K2),
                    STAT=IZERO)
                CALL ChkMemErr('READ_VENT','V_EDDY',IZERO)
                ALLOCATE(VT%W_EDDY(VT%I1+1:VT%I2,VT%K1+1:VT%K2),
                    STAT=IZERO)
                CALL ChkMemErr('READ_VENT','W_EDDY',IZERO)
            CASE(3)
                ALLOCATE(VT%U_EDDY(VT%I1+1:VT%I2,VT%J1+1:VT%J2),
                    STAT=IZERO)
                CALL ChkMemErr('READ_VENT','U_EDDY',IZERO)
                ALLOCATE(VT%V_EDDY(VT%I1+1:VT%I2,VT%J1+1:VT%J2),
                    STAT=IZERO)
                CALL ChkMemErr('READ_VENT','V_EDDY',IZERO)
                ALLOCATE(VT%W_EDDY(VT%I1+1:VT%I2,VT%J1+1:VT%J2),
                    STAT=IZERO)
                CALL ChkMemErr('READ_VENT','W_EDDY',IZERO)
        END SELECT
        ALLOCATE(VT%X_EDDY(VT%N_EDDY),STAT=IZERO)
        CALL ChkMemErr('READ_VENT','X_EDDY',IZERO)
        ALLOCATE(VT%Y_EDDY(VT%N_EDDY),STAT=IZERO)
        CALL ChkMemErr('READ_VENT','Y_EDDY',IZERO)
        ALLOCATE(VT%Z_EDDY(VT%N_EDDY),STAT=IZERO)
        CALL ChkMemErr('READ_VENT','Z_EDDY',IZERO)
        ALLOCATE(VT%CU_EDDY(VT%N_EDDY),STAT=IZERO)
        CALL ChkMemErr('READ_VENT','CU_EDDY',IZERO)
        ALLOCATE(VT%CV_EDDY(VT%N_EDDY),STAT=IZERO)
        CALL ChkMemErr('READ_VENT','CV_EDDY',IZERO)
        ALLOCATE(VT%CW_EDDY(VT%N_EDDY),STAT=IZERO)
        CALL ChkMemErr('READ_VENT','CW_EDDY',IZERO)
        VT%U_EDDY=0._EB
        VT%V_EDDY=0._EB
        VT%W_EDDY=0._EB
        VT%X_EDDY=0._EB
        VT%Y_EDDY=0._EB
        VT%Z_EDDY=0._EB
        VT%CU_EDDY=0._EB
        VT%CV_EDDY=0._EB
        VT%CW_EDDY=0._EB
    ENDIF EDDY_IF
ENDDO VENT_LOOP

M%WALL%TW = T_BEGIN

NOT_EVAC_IF_2: IF (.NOT.EVACUATION_ONLY(NM)) THEN
    M%WALL%EW = 0._EB
    M%WALL%KW = 0._EB
```

```
   DO IW=1,N_TOTAL_WALL_CELLS
      ALLOCATE(M%WALL(IW)%RHODW(N_TRACKED_SPECIES),STAT=IZERO)
      CALL ChkMemErr('INIT','WALL(IW)%RHODW',IZERO)
      M%WALL(IW)%RHODW = 0.1_EB ! Do not initialize to zero to
         avoid divide by zero in the first time step
   ENDDO
   M%WALL%AREA_ADJUST = 1._EB
   DO IW=1,N_TOTAL_WALL_CELLS
      ALLOCATE(M%WALL(IW)%ONE_D%MASSFLUX(0:N_TRACKED_SPECIES),
         STAT=IZERO)
      CALL ChkMemErr('INIT','WALL(IW)%ONE_D%MASSFLUX',IZERO)
      M%WALL(IW)%ONE_D%MASSFLUX = 0._EB
      ALLOCATE(M%WALL(IW)%ONE_D%MASSFLUX_ACTUAL(0:
         N_TRACKED_SPECIES),STAT=IZERO)
      CALL ChkMemErr('INIT','WALL(IW)%ONE_D%MASSFLUX_ACTUAL',
         IZERO)
      M%WALL(IW)%ONE_D%MASSFLUX_ACTUAL = 0._EB
   ENDDO
   M%WALL%NPPCW = 1
   M%WALL%BACK_INDEX = 0
   M%WALL%AW = 0._EB
   M%WALL%RAW = 0._EB
ENDIF NOT_EVAC_IF_2

M%WALL%RDN = 1._EB
M%WALL%UW0 = 0._EB
M%WALL%UW = 0._EB
M%WALL%UWS = 0._EB
M%WALL%OBST_INDEX = 0
M%WALL%VENT_INDEX = 0
ALLOCATE(M%DUWDT(N_TOTAL_WALL_CELLS),STAT=IZERO)
CALL ChkMemErr('INIT','DUWDT',IZERO)
M%DUWDT = 0._EB
ALLOCATE(M%PRESSURE_BC_INDEX (0:M%N_EXTERNAL_WALL_CELLS),STAT=
   IZERO)
CALL ChkMemErr('INIT','PRESSURE_BC_INDEX ',IZERO)
M%PRESSURE_BC_INDEX = NEUMANN
M%WALL%PRESSURE_BC_INDEX = NEUMANN
M%WALL%SURF_INDEX_ORIG = 0
M%WALL%BOUNDARY_TYPE = NULL_BOUNDARY
ALLOCATE(M%WALL_INDEX(0:M%CELL_COUNT,-3:3),STAT=IZERO)
CALL ChkMemErr('INIT','WALL_INDEX',IZERO)
M%WALL_INDEX = 0
```

```
ALLOCATE(M%EDGE_INDEX(0:M%CELL_COUNT,1:12),STAT=IZERO)
CALL ChkMemErr('INIT','EDGE_INDEX',IZERO)
M%EDGE_INDEX = 0
ALLOCATE(M%UVW_GHOST(0:M%CELL_COUNT,3),STAT=IZERO)
CALL ChkMemErr('INIT','UVW_GHOST',IZERO)
M%UVW_GHOST = 0

! Surface soot array

IF (N_SURFACE_DENSITY_SPECIES > 0 .AND. .NOT.EVACUATION_ONLY(NM)) THEN
   ALLOCATE(M%AWM_AEROSOL(N_TOTAL_WALL_CELLS,N_SURFACE_DENSITY_SPECIES),STAT=IZERO)
   CALL ChkMemErr('INIT','AWM_AEROSOL',IZERO)
   M%AWM_AEROSOL = 0._EB
   DO IW=1,N_TOTAL_WALL_CELLS
      ALLOCATE(M%WALL(IW)%AWM_AEROSOL(N_SURFACE_DENSITY_SPECIES),STAT=IZERO)
      CALL ChkMemErr('INIT','WALL(IW)%AWM_AEROSOL',IZERO)
      M%WALL(IW)%AWM_AEROSOL = 0._EB
   ENDDO
ENDIF

! Surface water arrays

IF (ACCUMULATE_WATER .AND. .NOT.EVACUATION_ONLY(NM)) THEN
   DO IW = 1,N_TOTAL_WALL_CELLS
      ALLOCATE(M%WALL(IW)%A_LP_MPUA(N_LP_ARRAY_INDICES),STAT=IZERO)
      CALL ChkMemErr('INIT','WALL(IW)%A_LP_MPUA',IZERO)
      M%WALL(IW)%A_LP_MPUA = 0._EB
   ENDDO
ENDIF

IF (.NOT.EVACUATION_ONLY(NM)) THEN

   DO IW = 1,N_TOTAL_WALL_CELLS
      ALLOCATE(M%WALL(IW)%LP_MPUA(N_LP_ARRAY_INDICES),STAT=IZERO)
      CALL ChkMemErr('INIT','WALL(IW)%LP_MPUA',IZERO)
      M%WALL(IW)%LP_MPUA = 0._EB
      ALLOCATE(M%WALL(IW)%LP_CPUA(N_LP_ARRAY_INDICES),STAT=IZERO)
      CALL ChkMemErr('INIT','WALL(IW)%LP_CPUA',IZERO)
      M%WALL(IW)%LP_CPUA = 0._EB
   ENDDO
ENDIF

! Surface work arrays
ALLOCATE(M%WALL_WORK1(N_TOTAL_WALL_CELLS),STAT=IZERO)
CALL ChkMemErr('INIT','WALL_WORK1',IZERO)
ALLOCATE(M%WALL_WORK2(N_TOTAL_WALL_CELLS),STAT=IZERO)
CALL ChkMemErr('INIT','WALL_WORK2',IZERO)

! Vegetation surface drag

ALLOCATE(M%VEG_DRAG(0:IBP1,0:JBP1),STAT=IZERO)
CALL ChkMemErr('INIT','VEG_DRAG',IZERO)
M%VEG_DRAG = 0._EB

! Set up boundary arrays for external boundaries of the current
    mesh

IWE = 0
IWG = M%N_EXTERNAL_WALL_CELLS + M%N_INTERNAL_WALL_CELLS + M%
   N_VIRTUAL_WALL_CELLS

DO K=0,KBP1
   DO J=0,JBP1
      I   = 0
      SURF_INDEX = DEFAULT_SURF_INDEX
      IOR = 1
      IF (J==0 .OR. J==JBP1 .OR. K==0 .OR. K==KBP1) THEN
         IWG = IWG + 1
         IW  = IWG
      ELSE
         IWE = IWE + 1
         IW  = IWE
      ENDIF
      CALL INIT_WALL_CELL(NM,I,J,K,0,IW,IOR,SURF_INDEX,IERR)
      IF (IERR>0) RETURN
   ENDDO
ENDDO
DO K=0,KBP1
   DO J=0,JBP1
      I   = IBP1
```

```
            SURF_INDEX = DEFAULT_SURF_INDEX
            IOR = -1
            IF (J==0 .OR. J==JBP1 .OR. K==0 .OR. K==KBP1) THEN
                IWG = IWG + 1
                IW  = IWG
            ELSE
                IWE = IWE + 1
                IW  = IWE
            ENDIF
            CALL INIT_WALL_CELL(NM,I,J,K,0,IW,IOR,SURF_INDEX,IERR)
            IF (IERR>0) RETURN
        ENDDO
    ENDDO

    DO K=0,KBP1
        DO I=0,IBP1
            J    = 0
            SURF_INDEX = DEFAULT_SURF_INDEX
            IOR = 2
            IF (I==0 .OR. I==IBP1 .OR. K==0 .OR. K==KBP1) THEN
                IWG = IWG + 1
                IW  = IWG
            ELSE
                IWE = IWE + 1
                IW  = IWE
            ENDIF
            CALL INIT_WALL_CELL(NM,I,J,K,0,IW,IOR,SURF_INDEX,IERR)
            IF (IERR>0) RETURN
        ENDDO
    ENDDO
    DO K=0,KBP1
        DO I=0,IBP1
            J    = JBP1
            SURF_INDEX = DEFAULT_SURF_INDEX
            IOR = -2
            IF (I==0 .OR. I==IBP1 .OR. K==0 .OR. K==KBP1) THEN
                IWG = IWG + 1
                IW  = IWG
            ELSE
                IWE = IWE + 1
                IW  = IWE
            ENDIF
            CALL INIT_WALL_CELL(NM,I,J,K,0,IW,IOR,SURF_INDEX,IERR)
```

```
         IF (IERR>0) RETURN
      ENDDO
   ENDDO

   IF (.NOT.EVACUATION_ONLY(NM)) THEN
   DO J=0,JBP1
      DO I=0,IBP1
         K    = 0
         SURF_INDEX = DEFAULT_SURF_INDEX
         IOR = 3
         IF (I==0 .OR. I==IBP1 .OR. J==0 .OR. J==JBP1) THEN
            IWG = IWG + 1
            IW  = IWG
         ELSE
            IWE = IWE + 1
            IW  = IWE
         ENDIF
         CALL INIT_WALL_CELL(NM,I,J,K,0,IW,IOR,SURF_INDEX,IERR)
         IF (IERR>0) RETURN
      ENDDO
   ENDDO
   DO J=0,JBP1
      DO I=0,IBP1
         K    = KBP1
         SURF_INDEX = DEFAULT_SURF_INDEX
         IOR = -3
         IF (I==0 .OR. I==IBP1 .OR. J==0 .OR. J==JBP1) THEN
            IWG = IWG + 1
            IW  = IWG
         ELSE
            IWE = IWE + 1
            IW  = IWE
         ENDIF
         CALL INIT_WALL_CELL(NM,I,J,K,0,IW,IOR,SURF_INDEX,IERR)
         IF (IERR>0) RETURN
      ENDDO
   ENDDO
   ENDIF

   ! Go through all obstructions and decide which cell faces ought
      to be given a wall cell index and initialized

   M%N_INTERNAL_WALL_CELLS = 0
```

```

OBST_LOOP_2: DO N=1,M%N_OBST
   OB=>M%OBSTRUCTION(N)

   DO K=OB%K1+1,OB%K2
      DO J=OB%J1+1,OB%J2
         I = OB%I1+1
         IF (I==1) CYCLE   ! Don't assign wall cell index to
             obstruction face pointing out of the computational
             domain
         IC = M%CELL_INDEX(I-1,J,K)
         IF (M%SOLID(IC) .AND. .NOT.M%OBSTRUCTION(M%
             OBST_INDEX_C(IC))%REMOVABLE) CYCLE   ! Permanently
             covered face
         IOR = -1
         SURF_INDEX = OB%SURF_INDEX(IOR)
         IW  = M%WALL_INDEX(IC,-IOR)
         IF (IW==0) THEN
            M%N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS +
                1
            IW  = M%N_EXTERNAL_WALL_CELLS + M%
               N_INTERNAL_WALL_CELLS
         ENDIF
         CALL INIT_WALL_CELL(NM,I,J,K,N,IW,IOR,SURF_INDEX,IERR)
         IF (IERR>0) RETURN
      ENDDO
   ENDDO

   DO K=OB%K1+1,OB%K2
      DO J=OB%J1+1,OB%J2
         I = OB%I2
         IF (I==M%IBAR) CYCLE  ! Don't assign wall cell index
             to obstruction face pointing out of the
             computational domain
         IC = M%CELL_INDEX(I+1,J,K)
         IF (M%SOLID(IC) .AND. .NOT.M%OBSTRUCTION(M%
             OBST_INDEX_C(IC))%REMOVABLE) CYCLE   ! Permanently
             covered face
         IOR = 1
         SURF_INDEX = OB%SURF_INDEX(IOR)
         IW  = M%WALL_INDEX(IC,-IOR)
         IF (IW==0) THEN
```

```
            M%N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS +
                 1
            IW   = M%N_EXTERNAL_WALL_CELLS + M%
                N_INTERNAL_WALL_CELLS
         ENDIF
         CALL INIT_WALL_CELL(NM,I,J,K,N,IW,IOR,SURF_INDEX,IERR)
         IF (IERR>0) RETURN
      ENDDO
   ENDDO

   DO K=OB%K1+1,OB%K2
      DO I=OB%I1+1,OB%I2
         J = OB%J1+1
         IF (J==1) CYCLE   ! Don't assign wall cell index to
              obstruction face pointing out of the computational
              domain
         IC = M%CELL_INDEX(I,J-1,K)
         IF (M%SOLID(IC) .AND. .NOT.M%OBSTRUCTION(M%
              OBST_INDEX_C(IC))%REMOVABLE) CYCLE   ! Permanently
              covered face
         IOR = -2
         SURF_INDEX = OB%SURF_INDEX(IOR)
         IW   = M%WALL_INDEX(IC,-IOR)
         IF (IW==0) THEN
            M%N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS +
                 1
            IW   = M%N_EXTERNAL_WALL_CELLS + M%
                N_INTERNAL_WALL_CELLS
         ENDIF
         CALL INIT_WALL_CELL(NM,I,J,K,N,IW,IOR,SURF_INDEX,IERR)
         IF (IERR>0) RETURN
      ENDDO
   ENDDO

   DO K=OB%K1+1,OB%K2
      DO I=OB%I1+1,OB%I2
         J = OB%J2
         IF (J==M%JBAR) CYCLE  ! Don't assign wall cell index
              to obstruction face pointing out of the
              computational domain
         IC = M%CELL_INDEX(I,J+1,K)
         IF (M%SOLID(IC) .AND. .NOT.M%OBSTRUCTION(M%
              OBST_INDEX_C(IC))%REMOVABLE) CYCLE   ! Permanently
```

```
                covered face
            IOR = 2
            SURF_INDEX = OB%SURF_INDEX(IOR)
            IW   = M%WALL_INDEX(IC,-IOR)
            IF (IW==0) THEN
               M%N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS +
                   1
               IW   = M%N_EXTERNAL_WALL_CELLS + M%
                  N_INTERNAL_WALL_CELLS
            ENDIF
            CALL INIT_WALL_CELL(NM,I,J,K,N,IW,IOR,SURF_INDEX,IERR)
            IF (IERR>0) RETURN
         ENDDO
      ENDDO

      DO J=OB%J1+1,OB%J2
         DO I=OB%I1+1,OB%I2
            K = OB%K1+1
            IF (K==1) CYCLE   ! Don't assign wall cell index to
                obstruction face pointing out of the computational
                domain
            IC = M%CELL_INDEX(I,J,K-1)
            IF (M%SOLID(IC)  .AND.  .NOT.M%OBSTRUCTION(M%
               OBST_INDEX_C(IC))%REMOVABLE) CYCLE   ! Permanently
                covered face
            IOR = -3
            SURF_INDEX = OB%SURF_INDEX(IOR)
            IW   = M%WALL_INDEX(IC,-IOR)
            IF (IW==0) THEN
               M%N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS +
                   1
               IW   = M%N_EXTERNAL_WALL_CELLS + M%
                  N_INTERNAL_WALL_CELLS
            ENDIF
            CALL INIT_WALL_CELL(NM,I,J,K,N,IW,IOR,SURF_INDEX,IERR)
            IF (IERR>0) RETURN
         ENDDO
      ENDDO

      DO J=OB%J1+1,OB%J2
         DO I=OB%I1+1,OB%I2
            K = OB%K2
            IF (K==M%KBAR) CYCLE  ! Don't assign wall cell index
                to obstruction face pointing out of the
                computational domain
            IC = M%CELL_INDEX(I,J,K+1)
            IF (M%SOLID(IC) .AND. .NOT.M%OBSTRUCTION(M%
                OBST_INDEX_C(IC))%REMOVABLE) CYCLE  ! Permanently
                covered face
            IOR = 3
            SURF_INDEX = OB%SURF_INDEX(IOR)
            IW   = M%WALL_INDEX(IC,-IOR)
            IF (IW==0) THEN
               M%N_INTERNAL_WALL_CELLS = M%N_INTERNAL_WALL_CELLS +
                   1
               IW   = M%N_EXTERNAL_WALL_CELLS + M%
                   N_INTERNAL_WALL_CELLS
            ENDIF
            CALL INIT_WALL_CELL(NM,I,J,K,N,IW,IOR,SURF_INDEX,IERR)
            IF (IERR>0) RETURN
         ENDDO
      ENDDO

ENDDO OBST_LOOP_2

!Initialize PSUM for zone cases

IF (N_ZONE > 0) THEN
   N_ZONE_LOOP: DO IPZ = 1,N_ZONE
      PSUM(IPZ,NM) = 0._EB
      IF (EVACUATION_ONLY(NM)) EXIT N_ZONE_LOOP
      DO K=1,M%KBAR
         DO J=1,M%JBAR
            DO I=1,M%IBAR
               IF (M%PRESSURE_ZONE(I,J,K) /= IPZ) CYCLE
               IF (M%SOLID(M%CELL_INDEX(I,J,K)))     CYCLE
               VC   = M%DX(I)*M%RC(I)*M%DY(J)*M%DZ(K)
               IF (N_TRACKED_SPECIES>0) ZZ_GET(1:
                   N_TRACKED_SPECIES) = M%ZZ(I,J,K,1:
                   N_TRACKED_SPECIES)
               CALL GET_SPECIFIC_HEAT(ZZ_GET,CP,M%TMP(I,J,K))
               RTRM = M%RSUM(I,J,K)/(CP*M%PBAR(K,IPZ))
               PSUM(IPZ,NM) = PSUM(IPZ,NM) + VC*(1._EB/M%PBAR(K
                   ,IPZ)-RTRM)
            ENDDO
            ENDDO
         ENDDO
      ENDDO N_ZONE_LOOP
ENDIF

! Set up wall cell arrays for VIRTUAL boundaries

IW = M%N_EXTERNAL_WALL_CELLS + M%N_INTERNAL_WALL_CELLS

DEVICE_LOOP: DO N=1,N_DEVC
   IF (EVACUATION_ONLY(NM)) CYCLE DEVICE_LOOP
   DV => DEVICE(N)
   IF (DV%MESH/=NM) CYCLE DEVICE_LOOP
   IF (DV%QUANTITY/='CABLE_TEMPERATURE') CYCLE DEVICE_LOOP
   IW = IW + 1
   DV%VIRTUAL_WALL_INDEX = IW
   I = DV%I
   J = DV%J
   K = DV%K
   SURF_INDEX = DV%SURF_INDEX
   CALL INIT_WALL_CELL(NM,I,J,K,0,IW,0,SURF_INDEX,IERR)
ENDDO DEVICE_LOOP

! Determine back wall index for exposed surfaces

DO IW=M%N_EXTERNAL_WALL_CELLS+1,M%N_EXTERNAL_WALL_CELLS+M%
   N_INTERNAL_WALL_CELLS
   IF (EVACUATION_ONLY(NM)) CYCLE
   ! Only assign BACK_INDEX to wall cells that are not attached
        to the exterior boundary of the computational domain
   SF=>SURFACE(M%WALL(IW)%SURF_INDEX)
   IF (SF%BACKING==EXPOSED) THEN
      II = M%WALL(IW)%II
      JJ = M%WALL(IW)%JJ
      KK = M%WALL(IW)%KK
      IC = M%CELL_INDEX(II,JJ,KK)
      IOR = M%WALL(IW)%IOR
      IF (.NOT.M%SOLID(IC)) M%WALL(IW)%BACK_INDEX = M%
         WALL_INDEX(IC,IOR)
      IF (      M%SOLID(IC)) THEN
         SELECT CASE(IOR)
            CASE(-1)
               II=II+1
```

```
                CASE( 1)
                    II=II-1
                CASE(-2)
                    JJ=JJ+1
                CASE( 2)
                    JJ=JJ-1
                CASE(-3)
                    KK=KK+1
                CASE( 3)
                    KK=KK-1
            END SELECT
            IC = M%CELL_INDEX(II,JJ,KK)
            M%WALL(IW)%BACK_INDEX = M%WALL_INDEX(IC,IOR)
        ENDIF
    ENDIF
ENDDO

! Set clocks and counters related to frequency of solid phase
    conduction updates

M%BC_CLOCK      = T_BEGIN
M%WALL_COUNTER = 0

! Set clock for boudary fuel vegetation model

M%VEG_CLOCK_BC = T_BEGIN

! Allocate arrays for storing velocity boundary condition info

N_EDGES_DIM = 4*(IBP1*JBP1+IBP1*KBP1+JBP1*KBP1)
IF (EVACUATION_ONLY(NM)) N_EDGES_DIM = 4*(IBP1*KBP1+JBP1*KBP1)
DO N=1,M%N_OBST
    OB=>M%OBSTRUCTION(N)
    IPTS = OB%I2-OB%I1
    JPTS = OB%J2-OB%J1
    KPTS = OB%K2-OB%K1
    IF (EVACUATION_ONLY(NM)) THEN
        N_EDGES_DIM = N_EDGES_DIM + 4*(IPTS*KPTS+JPTS*KPTS)
    ELSE
        N_EDGES_DIM = N_EDGES_DIM + 4*(IPTS*JPTS+IPTS*KPTS+JPTS*
            KPTS)
    ENDIF
ENDDO
```

```

ALLOCATE(M%IJKE(16,N_EDGES_DIM),STAT=IZERO)
CALL ChkMemErr('INIT','IJKE',IZERO)
M%IJKE   = 0
ALLOCATE(M%OME_E(0:N_EDGES_DIM,-2:2),STAT=IZERO)
CALL ChkMemErr('INIT','OME_E',IZERO)
M%OME_E = 0._EB
ALLOCATE(M%TAU_E(0:N_EDGES_DIM,-2:2),STAT=IZERO)
CALL ChkMemErr('INIT','TAU_E',IZERO)
M%TAU_E = 0._EB
ALLOCATE(M%EDGE_TYPE(N_EDGES_DIM,2),STAT=IZERO)
CALL ChkMemErr('INIT','EDGE_TYPE',IZERO)
M%EDGE_TYPE = SOLID_EDGE
ALLOCATE(M%EDGE_INTERPOLATION_FACTOR(N_EDGES_DIM,2),STAT=IZERO)
CALL ChkMemErr('INIT','EDGE_INTERPOLATION_FACTOR',IZERO)
M%EDGE_INTERPOLATION_FACTOR = 1._EB

! Initialize and allocate lagrangian particle/PARTICLE arrays

M%NLP = 0
M%NLPDIM = 1000
IF (PARTICLE_FILE .AND. .NOT.EVACUATION_ONLY(NM)) THEN
   ALLOCATE(M%LAGRANGIAN_PARTICLE(M%NLPDIM),STAT=IZERO)
   CALL ChkMemErr('INIT','PARTICLE',IZERO)
ENDIF

! Allocate array to hold character strings for Smokeview file

M%N_STRINGS       =   0
M%N_STRINGS_MAX = 100
ALLOCATE(M%STRING(M%N_STRINGS_MAX),STAT=IZERO)
CALL ChkMemErr('INIT','STRING',IZERO)

! Set up arrays to hold velocity boundary condition info

CALL INITIALIZE_EDGES

! Initialize Pressure solver

CALL INITIALIZE_POISSON_SOLVER
IF (IERR/=0) RETURN

```

```
! Determine which wall cells to assign for solid phase
    thermocouples and profiles

CALL INITIALIZE_DEVC
IF (IERR/=0) RETURN

CALL INITIALIZE_PROF
IF (IERR/=0) RETURN

! Initialize Mesh Exchange

CALL INITIALIZE_INTERPOLATION


CONTAINS
```

Appendix C

Flame Height Calculation, post-prosessing

```
program compute_flame_height

character(30) :: infile(16,3)
real :: z(0:200),hrrpul(200),height(16,3),qstar(16),diameter,
    sumold
integer :: i,n,npts

diameter = 1.13   ! Equivalent diameter of 1 m2 square

qstar(1)   = 0.1
qstar(2)   = 0.2
qstar(3)   = 0.5
qstar(4)   = 1.0
qstar(5)   = 2.0
qstar(6)   = 5.0
qstar(7)   = 10.
qstar(8)   = 20.
qstar(9)   = 50.
qstar(10)  = 100.
qstar(11)  = 200.
qstar(12)  = 500.
qstar(13)  = 1000.
qstar(14)  = 2000.
qstar(15)  = 5000.
qstar(16)  = 10000.

infile(1,1)  = 'Qs=p1_RI=05_fds2ascii.csv'
infile(2,1)   = 'Qs=p2_RI=05_fds2ascii.csv'
infile(3,1)   = 'Qs=p5_RI=05_fds2ascii.csv'
infile(4,1)   = 'Qs=1_RI=05_fds2ascii.csv'
infile(5,1)   = 'Qs=2_RI=05_fds2ascii.csv'
infile(6,1)   = 'Qs=5_RI=05_fds2ascii.csv'
infile(7,1)   = 'Qs=10_RI=05_fds2ascii.csv'
infile(8,1)   = 'Qs=20_RI=05_fds2ascii.csv'
infile(9,1)   = 'Qs=50_RI=05_fds2ascii.csv'
infile(10,1) = 'Qs=100_RI=05_fds2ascii.csv'
infile(11,1) = 'Qs=200_RI=05_fds2ascii.csv'
infile(12,1) = 'Qs=500_RI=05_fds2ascii.csv'
infile(13,1) = 'Qs=1000_RI=05_fds2ascii.csv'
infile(14,1) = 'Qs=2000_RI=05_fds2ascii.csv'
infile(15,1) = 'Qs=5000_RI=05_fds2ascii.csv'
infile(16,1) = 'Qs=10000_RI=05_fds2ascii.csv'

infile(1,2)   = 'Qs=p1_fds2ascii.csv'
infile(2,2)   = 'Qs=p2_fds2ascii.csv'
infile(3,2)   = 'Qs=p5_fds2ascii.csv'
infile(4,2)   = 'Qs=1_fds2ascii.csv'
infile(5,2)   = 'Qs=2_fds2ascii.csv'
infile(6,2)   = 'Qs=5_fds2ascii.csv'
infile(7,2)   = 'Qs=10_fds2ascii.csv'
infile(8,2)   = 'Qs=20_fds2ascii.csv'
infile(9,2)   = 'Qs=50_fds2ascii.csv'
infile(10,2) = 'Qs=100_fds2ascii.csv'
infile(11,2) = 'Qs=200_fds2ascii.csv'
infile(12,2) = 'Qs=500_fds2ascii.csv'
infile(13,2) = 'Qs=1000_fds2ascii.csv'
infile(14,2) = 'Qs=2000_fds2ascii.csv'
infile(15,2) = 'Qs=5000_fds2ascii.csv'
infile(16,2) = 'Qs=10000_fds2ascii.csv'

infile(1,3)   = 'Qs=p1_RI=20_fds2ascii.csv'
infile(2,3)   = 'Qs=p2_RI=20_fds2ascii.csv'
infile(3,3)   = 'Qs=p5_RI=20_fds2ascii.csv'
infile(4,3)   = 'Qs=1_RI=20_fds2ascii.csv'
infile(5,3)   = 'Qs=2_RI=20_fds2ascii.csv'
infile(6,3)   = 'Qs=5_RI=20_fds2ascii.csv'
infile(7,3)   = 'Qs=10_RI=20_fds2ascii.csv'
infile(8,3)   = 'Qs=20_RI=20_fds2ascii.csv'
infile(9,3)   = 'Qs=50_RI=20_fds2ascii.csv'
infile(10,3) = 'Qs=100_RI=20_fds2ascii.csv'
infile(11,3) = 'Qs=200_RI=20_fds2ascii.csv'
infile(12,3) = 'Qs=500_RI=20_fds2ascii.csv'
infile(13,3) = 'Qs=1000_RI=20_fds2ascii.csv'
infile(14,3) = 'Qs=2000_RI=20_fds2ascii.csv'
infile(15,3) = 'Qs=5000_RI=20_fds2ascii.csv'
infile(16,3) = 'Qs=10000_RI=20_fds2ascii.csv'

write(6,"(a)") "Q*,L/D (RI=5),L/D (RI=10),L/D (RI=20)"

file_loop: do n=1,16

   resolution_loop: do i=1,3

      if (i==1) npts=39
      if (i==2) npts=76
      if (i==3) npts=151

      open(10,file=infile(n,i),form='formatted',status='old')
      read(10,*)
      read(10,*)
      z    = 0.
      z(0) = 0.
      do k=1,npts
         read(10,*,end=20) z(k),hrrpul(k)
      enddo
   20 continue
      sum = 0.
      do k=1,npts
    !!   sum = sum + hrrpul(k)*(z(k)-z(k-1))
         sum = sum + hrrpul(k)
      enddo
      sum1 = 0.
      do k=1,npts
    !!   sum1 = sum1 + hrrpul(k)*(z(k)-z(k-1))
         sumold = sum1
         sum1 = sum1 + hrrpul(k)
         if (sum1/sum>0.99) then
            height(n,i) = z(k-1) + (z(k)-z(k-1))*(0.99*sum-
               sumold)/(sum1-sumold)
            exit
         endif
      enddo

    enddo resolution_loop

    write(6,"(f8.1,',',f8.2,',',f8.2,',',f8.2)") qstar(n),height
        (n,1)/diameter,height(n,2)/diameter,height(n,3)/diameter
    close(10)

enddo file_loop

end program
```