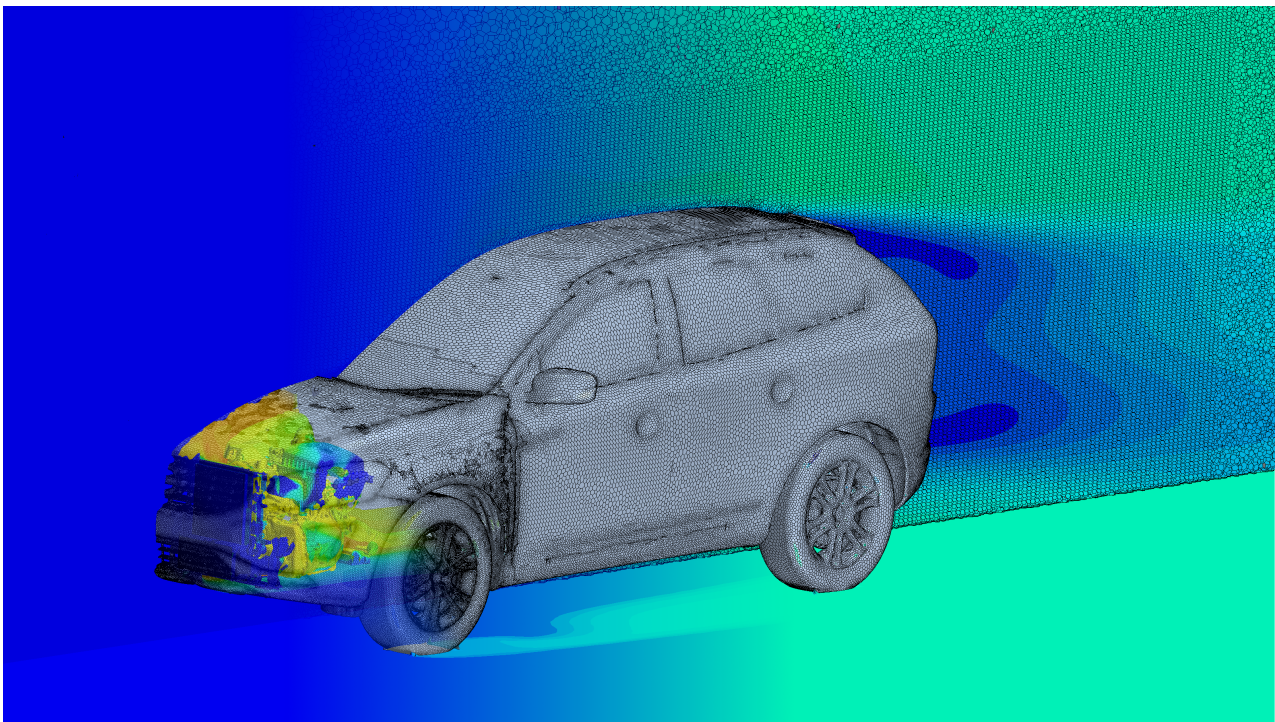


CHALMERS



CFD for Underhood Modeling

Development of an Efficient Method

Master's Thesis in Fluid Mechanics

EMIL LJUNGSKOG

ULF NILSSON

Department of Applied Mechanics

Division of Fluid Dynamics

CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2014

Master's Thesis 2014:37

MASTER'S THESIS IN FLUID MECHANICS

CFD for Underhood Modeling

Development of an Efficient Method

EMIL LJUNGSKOG
ULF NILSSON

Department of Applied Mechanics
Division of Fluid Dynamics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2014

CFD for Underhood Modeling
Development of an Efficient Method
EMIL LJUNGSKOG
ULF NILSSON

© EMIL LJUNGSKOG , ULF NILSSON, 2014

Master's Thesis 2014:37
ISSN 1652-8557
Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:

Temperature and velocity field on a cut plane through the computational domain, with temperature profile on the engine. The surface and volume meshes are also depicted. The car is a Volvo XC60.

Chalmers Reproservice
Göteborg, Sweden 2014

CFD for Underhood Modeling
Development of an Efficient Method
Master's Thesis in Fluid Mechanics
EMIL LJUNGSKOG
ULF NILSSON
Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology

ABSTRACT

Today's development of cooling systems for passenger cars relies on one-dimensional system simulations, which are heavily dependent on accurate input data obtained from testing or from CFD. Current three-dimensional CFD methods are deemed too computational and workload expensive to be used, which is why the present work was aimed at investigating the possibilities to develop a CFD model of the complete underhood compartment, with a high degree of automation.

In the present work, two methods for CFD analysis of underhood flow in passenger cars were developed and compared. Both methods used ANSA for geometry cleanup and STAR-CCM+ as CFD solver, while the volume meshing differed. Method A used a polyhedral mesh created in STAR-CCM+, while a hexahedral mesh was used in method B. Surface wrapping was extensively used in both methods to reduce the manual workload in the geometry cleanup step.

In order to validate the methods, they were both applied to a validation case. Both methods underestimated the heat rejection in the radiator with approximately 13 % compared to wind tunnel measurements, while the heat rejection in the Charge Air Cooler was underestimated by 4.5 % and 6.0 % in method A and B, respectively. However, there was a large uncertainty in the experimental results, why further research will be needed to determine the validity of the developed methods.

It was concluded that method A showed a larger potential to be used in production cases due to its lower time consumption and superior convergence behavior, even though more work is needed before it can be fully incorporated into the work flow. The time reduction compared to similar methods currently in use at Volvo cars was estimated to roughly 30 %.

Keywords: Computational Fluid Dynamics, CFD, underhood flow, heat exchanger modeling, MRF

PREFACE

The report you are reading is the result of a project carried out during the spring of 2014 at the Outer Cooling System group at Volvo Cars, aimed at investigating the possibilities of developing an efficient method for performing underhood flow simulations.

The work was supervised by Cooling System CAE Analyst Anders Wedin at Volvo, and examined by Professor Lars Davidson at the Division of Fluid Dynamics at Chalmers University of Technology.

ACKNOWLEDGEMENTS

As the authors of this thesis, we would like to thank everyone that have contributed to this project. We would especially like to acknowledge the staff at the Outer Cooling System group at Volvo Cars, for hosting us during this project. A special thanks goes to our supervisor Anders Wedin, for his support and very successful efforts to make our time at Volvo both instructive and enjoyable. We would also like to thank Group Manager Stefan Molén for allocating all the resources needed for us to complete this project.

Furthermore, we would like to acknowledge the staff at the Thermodynamic CFD department at Volvo, who have been very helpful during the project.

We would also like to thank CD-adapco for providing licences for their CFD software suite STAR-CCM+, and in particular Jonathan Thompson for his invaluable support on how to use this software for our applications.

NOMENCLATURE

Abbreviations

CAC	Charge Air Cooler
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CFD	Computational Fluid Dynamics
FANS	Favre Averaged Navier-Stokes
LES	Large Eddy Simulations
MRF	Multiple Reference Frames
NURBS	Non-Uniform Rational B-Splines
PLM	Product Lifecycle Management
RANS	Reynolds Averaged Navier-Stokes
TSTE	Taylor Series Truncation Error
UDS	Upwind Differencing Scheme

Greek symbols

ε_{ijk}	Levi-Civita tensor
μ	Dynamic viscosity
ν	Kinematic viscosity
ν_t	Turbulent viscosity
ρ	Density
ε	Turbulent dissipation
τ_{ij}	Viscous stress tensor
δ_{ij}	Kronecker delta
$\langle \phi \rangle$	Time averaged quantity
$\tilde{\phi}$	Favre averaged quantity

Roman symbols

e	Specific internal energy
k	Turbulent kinetic energy
C_p	Specific heat capacity at constant pressure
C_v	Specific heat capacity at constant volume
n_i	Normal vector, i th component
p	Pressure
q_i	Heat flux vector, i th component
S_{ij}	Strain rate tensor
t	Time
T	Temperature
u_i	Velocity vector, i th component
x_i	Spatial coordinate, i th component

CONTENTS

Abstract	i
Preface	iii
Acknowledgements	iii
Nomenclature	v
Contents	vii
1 Introduction	1
1.1 Background	1
1.2 Purpose	1
1.3 Limitations	1
2 Theory	2
2.1 Governing equations	2
2.2 Turbulent flow	2
2.2.1 RANS	3
2.2.2 Realizable k - ϵ	4
2.3 Discretization methods	5
2.3.1 First order UDS	5
2.3.2 Second order UDS	5
2.4 Cooling system	6
2.4.1 Radiator	6
2.4.2 Charge air cooler	7
2.4.3 Condenser	7
2.4.4 Axial fan	7
2.5 Fan modeling	7
2.5.1 Single and multiple reference frames	7
2.6 Heat exchanger modeling	9
2.6.1 Porous media model	9
2.6.2 Single stream heat exchanger model	9
2.6.3 Dual stream heat exchanger model	10
2.7 Geometry representation	11
2.7.1 Geometry clean-up	12
2.8 Volume meshing	15
2.8.1 Hexahedral mesher	15
2.8.2 Polyhedral mesher	17
3 Method	18
3.1 CFD software	19
3.2 Geometry preparation	19
3.3 CAD cleanup	20
3.3.1 Method A	20
3.3.2 Method B	22
3.4 Volume meshing	22
3.4.1 Method A	24
3.4.2 Method B	25
3.5 Case setup	28
3.5.1 Model selection	28
3.5.2 Boundary conditions	28
3.5.3 Initial conditions	28
3.5.4 Physics	30

3.5.5 Solver settings	31
4 Results	32
4.1 Heat exchangers	32
4.1.1 Radiator	32
4.1.2 Charge air cooler	34
4.1.3 Condenser	35
4.2 MRF interfaces	36
4.2.1 Flow behavior	38
5 Discussion	39
5.1 Method comparison	39
5.2 Validation	40
5.2.1 Evaluation of the MRF model	40
6 Conclusions	41
References	42
A Harpoon batch script	I
B Matlab scripts	III
B.1 Porous media coefficients	III
B.2 Coolant density	V
C STAR-CCM+ script for interface creation	VI

1 Introduction

1.1 Background

Development of the cooling circuit for all Volvo cars are done at the outer cooling system group, where early concepts are evaluated using one-dimensional system simulations and simplified three-dimensional analysis. These types of simulations need a lot of input data, since they basically just link different objects together and simulate their combined response under different loads. The most important load case in the current context is a hill climb with a trailer weight of 1800 kg (HCTR1800). The components of the cooling system are dimensioned to manage a climb up a 6 % slope for a limited amount of time, and the experimental investigations are set to resemble this situation.

Many input parameters are obtained from tests and suppliers, while others are obtained using CFD or simpler methods. One such example is the mass flow through the front mounted heat exchangers, which is obtained from a very simplified three-dimensional model of the engine bay combined with one-dimensional flow analysis. This method is considered inaccurate, but the alternative approaches are deemed too expensive; both in the computational and workload sense. Considering this, it is found that a need for a new method has arisen.

1.2 Purpose

The purpose of this thesis is to investigate the possibility to develop a method for obtaining a CFD model of the complete underhood compartment of a car, with a high degree of automation. Since the method will be used in early concept stages for evaluation of different designs, the focus is not mainly accuracy, but rather ease of use and automation. In order for the method to be useful at the outer cooling systems department, the amount of manual work has to be kept at an absolute minimum.

1.3 Limitations

The project is limited in time to 20 weeks for two MSc students. Computer resources are limited to those available at VCC; both hardware and software. Furthermore, only steady-state solutions of one load case for one model are considered. Only the heat transfer occurring in the cooling pack will be considered, while radiation and air humidity will be discarded.

2 Theory

This chapter will briefly introduce the theoretic aspects of this thesis. A description of the equations governing fluid flow will be followed by a short introduction to turbulence modeling and discretization methods. However, the main focus will be on presenting the structure of the cooling system, and the theory on how to model it in an underhood flow analysis. Furthermore, a discussion on geometric surface representation is followed by an introduction to different mesh types and how they can be generated.

2.1 Governing equations

The governing equations describing a compressible, Newtonian fluid are three in number: the continuity equation (2.1), the momentum equation (2.2) and the energy equation (2.5). The continuity equation stems from a mass balance over a fluid element and can be written as [1]

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i} (\rho u_i) = 0. \quad (2.1)$$

The momentum equation, which relates the forces acting on a fluid element to the motion of the fluid according to Newton's second law, reads [1]

$$\frac{\partial}{\partial t} (\rho u_i) + \frac{\partial}{\partial x_j} (\rho u_i u_j) = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho f_i, \quad (2.2)$$

where ρf_i are body forces. The viscous stresses, τ_{ij} , are defined according to the constitutive law of a Newtonian fluid:

$$\tau_{ij} = \left(2\mu s_{ij} - \frac{2}{3}\mu \frac{\partial u_k}{\partial x_k} \delta_{ij} \right), \quad (2.3)$$

with the strain rate tensor

$$s_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right). \quad (2.4)$$

The energy equation, formulated in terms of the specific total energy, $e = e_{internal} + \frac{1}{2}u_i u_i$, reads [2, 3]

$$\frac{\partial}{\partial t} (\rho e) + \frac{\partial}{\partial x_i} (\rho e u_i) = -\frac{\partial}{\partial x_i} (p u_i) + \frac{\partial}{\partial x_i} (\tau_{ij} u_j) + \frac{\partial q_i}{\partial x_i} + \rho u_i f_i + S_e, \quad (2.5)$$

where S_e is an energy source term and q_i denotes the heat flux, often modeled using Fourier's law. Additional equations of state are used to couple some of the unknown thermodynamic quantities according to

$$p = p(\rho, T), \quad (2.6)$$

$$e = e(\rho, T). \quad (2.7)$$

In the case of an ideal gas, the state equations read [1]

$$p = \rho R T, \quad (2.8)$$

$$e = C_v T, \quad (2.9)$$

where ρ , R and C_v are the density of the gas, the ideal gas constant, and the specific heat capacity at constant volume, respectively.

2.2 Turbulent flow

For most engineering applications, it is often too computationally demanding to resolve all turbulent scales. One way to reduce the computational resources needed is averaging. Averaging the flow variables in time, decomposing the instantaneous velocity and pressure into one steady and one fluctuating part according to

$$u_i = \langle u_i \rangle + u'_i, \quad (2.10)$$

$$p = \langle p \rangle + p', \quad (2.11)$$

and inserting the result into the governing equations yields the time averaged Navier-Stokes equations, also known as Reynolds Averaged Navier-Stokes (RANS). Here, $\langle \cdot \rangle$ and $'$ denote the time averaged and fluctuating parts, respectively. In other modeling methods different averaging approaches are used. A volume averaging is for example used in the Large Eddy Simulation (LES) approach, resulting in a method modeling the smallest turbulent scales, but resolving eddies larger than the grid size [4]. A more accurate transient solution is achieved, but at a greater cost.

There exists a range of different methods described in the literature, with varying accuracy and expense. RANS is the important one in the current context and will be covered in more detail in this section.

2.2.1 RANS

The time averaged part of a quantity $\phi(x_i, t)$ is calculated as

$$\langle \phi(x_i, t) \rangle = \frac{1}{T} \int_{t-T/2}^{t+T/2} \phi(x_i, \tau) d\tau, \quad (2.12)$$

where the period T is chosen to fulfill statistical requirements. This is the approach usually applied when the flow can be considered to be incompressible. However, to reduce the number of separate terms in need of modeling when the flow is compressible a different averaging methodology, known as Favre averaging, is often used. In this method the mean is defined as [1]

$$\tilde{\phi}(x_i, t) = \frac{1}{\langle \rho \rangle T} \int_{t-T/2}^{t+T/2} \rho(x_i, \tau) \phi(x_i, \tau) d\tau = \frac{\langle \rho \phi \rangle}{\langle \rho \rangle}. \quad (2.13)$$

It can be seen from the definition above that the instantaneous velocity can be written similarly to (2.10) as

$$u_i = \tilde{u}_i + u''_i, \quad (2.14)$$

where u'' now includes both turbulent velocity and density fluctuations.

Using this density-weighted averaging procedure to decompose the velocity and the specific total energy e before time averaging reduces the governing equations to the Favre Averaged Navier-Stokes (FANS) equations [1, 3]

$$\frac{\partial \langle \rho \rangle}{\partial t} + \frac{\partial}{\partial x_i} (\langle \rho \rangle \tilde{u}_i) = 0, \quad (2.15)$$

$$\frac{\partial}{\partial t} (\langle \rho \rangle \tilde{u}_i) + \frac{\partial}{\partial x_j} (\langle \rho \rangle \tilde{u}_i \tilde{u}_j) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j} \left(\langle \tau_{ij} \rangle - \underbrace{\langle \rho u''_i u''_j \rangle}_i \right), \quad (2.16)$$

$$\begin{aligned} \frac{\partial}{\partial t} (\langle \rho \rangle \tilde{E}) + \frac{\partial}{\partial x_j} (\langle \rho \rangle \tilde{u}_j \tilde{E}) = & -\frac{\partial}{\partial x_j} (\langle p \rangle \tilde{u}_j) \\ & + \frac{\partial}{\partial x_j} \left((\langle \tau_{ji} \rangle - \underbrace{\langle \rho u''_j u''_i \rangle}_i) \tilde{u}_i \right) \\ & - \frac{\partial}{\partial x_j} \left(\langle q_j \rangle + \underbrace{\left\langle \rho u''_j \left(e_{internal} + \frac{p}{\rho} \right)'' \right\rangle}_{ii} \right) \\ & + \frac{\partial}{\partial x_j} \left(\underbrace{-\frac{1}{2} \langle \rho u''_j u''_i u''_i \rangle + \langle \tau_{ji} u''_i \rangle}_{iii} \right). \end{aligned} \quad (2.17)$$

Here the energy source term has been left out for simplicity. The term τ_{ij} is defined according to (2.3) while the specific total energy e consist of contributions from both internal and kinetic energy, $e = e_{internal} + \frac{1}{2} u_i u_i$.

It can be noted that additional, unknown terms representing correlations between the fluctuating quantities have been introduced, which need to be modeled. In the above equation system the terms i – iii are unknowns corresponding to the following physical phenomena:

- i Turbulent (Reynolds) stresses.
- ii Reynolds heat flux.
- iii Turbulent transport and work.

Turbulence modeling provides a solution to the closure problem of the open system of equations presented above by modeling unknowns in terms of mean flow quantities. The focus in the remainder of this section will be on describing how the stresses are modeled. The reader is referred to the documentation written by Todd A. Oliver [3] for more information about the remaining terms, that are deemed to be out of the scope of this thesis.

2.2.2 Realizable k - ε

The basic concept of the realizable k - ε model will be introduced using an incompressible approach. Exact transport equations for the Reynolds stresses can be derived from the Navier-Stokes equation. However, eddy-viscosity models are often used, introducing an eddy (turbulent) viscosity according to the Boussinesq assumption

$$\langle v'_i v'_j \rangle = -\nu_t \left(\frac{\partial \langle v_i \rangle}{\partial x_j} + \frac{\partial \langle v_j \rangle}{\partial x_i} \right) + \frac{2}{3} \delta_{ij} k, \quad (2.18)$$

where k is the turbulent kinetic energy and ν_t is the turbulent viscosity. In two-equation approaches, ν_t is modeled using two quantities, e.g. for the k - ε turbulence model:

$$\nu_t = c_\mu \frac{k^2}{\varepsilon}, \quad (2.19)$$

where ε is the turbulent dissipation. In the standard k - ε model, c_μ is a constant determined from experimental results. However, the value of the constant can differ significantly with varying flow conditions, introducing errors for models using a constant value of the coefficient in some flow cases.

The realizable k - ε model tries to remedy the poor prediction of the turbulent length scale and viscosity in some type of flows by proposing new formulations of (2.19) and the modeled transport equation of the turbulent dissipation [5]. The proposed expressions are developed to ensure realizability of the normal Reynolds stresses and that Schwartz's inequality is not violated.

As opposed to the standard model, which derives a modeled equation for the dissipation using the k -equation as a starting point, the realizable model's equation originates from the exact transport equation of $\langle \omega'_i \omega'_i \rangle$, where ω'_i is the fluctuating vorticity. After appropriate modeling one readily obtains the standard k -equation and a modeled equation for ε . This new equation for ε is more suitable for some of the flow conditions for which the standard model gives erroneous results, and reads [5]

$$\frac{\partial \varepsilon}{\partial t} + \langle u_j \rangle \frac{\partial \varepsilon}{\partial x_j} = \frac{\partial}{\partial x_j} \left(\frac{\nu_t}{\sigma_\varepsilon} \frac{\partial \varepsilon}{\partial x_j} \right) + c_1 S \varepsilon - c_2 \frac{\varepsilon^2}{k + \sqrt{\nu \varepsilon}}. \quad (2.20)$$

Here S denotes the mean strain rate ($\sqrt{2S_{ij}S_{ij}}$) and ν is the kinematic viscosity while c_1 , c_2 and σ_ε are coefficients determined through experiments, defined as [5]

$$\begin{cases} c_1 &= \max \left(0.43, \frac{\eta}{5+\eta} \right) \quad , \quad \eta = \frac{\sqrt{2S_{ij}S_{ij}}k}{\varepsilon}, \\ c_2 &= 1.9, \\ \sigma_\varepsilon &= 1.0. \end{cases}$$

The only difference between the realizable and standard k - ε model in the modeled dissipation equation can be seen to be the source (production) term [5]. The expression for the constant c_μ also differs between the models. The standard model uses a constant value, while the realizable calculates it as [5]

$$c_\mu = \left(A_0 + A_S U^{(*)} \frac{k}{\varepsilon} \right)^{-1}, \quad (2.21)$$

where A_0 , A_S and $U^{(*)}$ are coefficients dependent on S_{ij} , Ω_{ij} and ω_i . For an exact definition the reader is referred to the work done by Shih et al. [5].

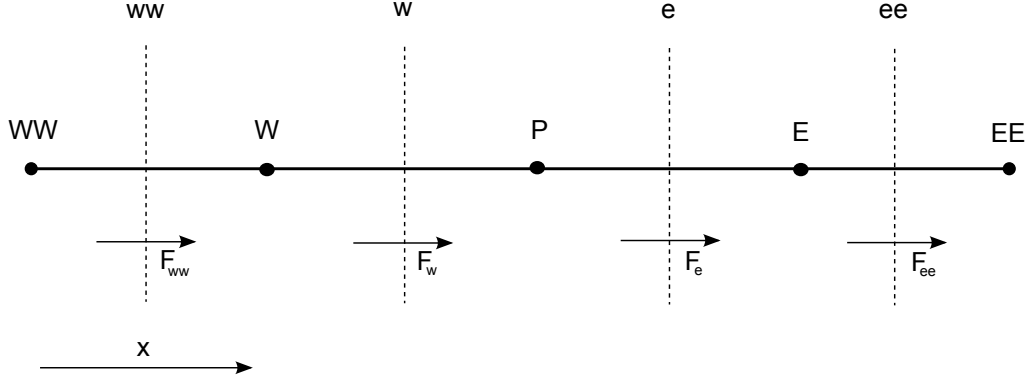


Figure 2.1: One dimensional discretized domain with indicated fluxes.

2.3 Discretization methods

Applying the finite volume method to the modeled equations and using the Gaussian theorem to rewrite the volume integrals to surface integrals as

$$\int_{CV} \frac{\partial \phi_i}{\partial x_i} dV = \int_{CS} \phi_i n_i dS, \quad (2.22)$$

gives rise to the need of evaluating the quantity ϕ_i at the boundary of the control volume. For instance, the convection term in (2.2) reduces to

$$\int_{CV} \frac{\partial}{\partial x_j} (\rho u_j \phi_i) dV = \sum_k \int_{A_k} \rho n_j u_j \phi_i dA_k, \quad (2.23)$$

where the index k runs over the faces of the control volume.

There exist a number of different differencing schemes to evaluate the values of the needed quantities at the face centers of the computational grid. Which scheme is most suitable is very case and term dependent and influence the accuracy and computational demand of the simulation. For convection-diffusion problems, Upwind Differencing Schemes (UDS) are often applied. It is at least conditionally bounded and possesses the transportiveness property, i.e. accounts for the direction of the flow [1]. This section includes a brief explanation of the first and second order UDS.

2.3.1 First order UDS

The first order UDS determines the value at the face depending on the flux across the interface. The value of ϕ_i at the face w , between node P and W , is evaluated as [1]

$$\phi_w = \begin{cases} \phi_W & F_w \geq 0, \\ \phi_P & F_w < 0. \end{cases} \quad (2.24)$$

Hence, considering the flow conditions depicted in figure 2.1 the following relation can be seen to hold for the adjacent faces to the node P

$$\phi_e = \phi_P, \quad \phi_w = \phi_W.$$

The main drawback of the first order UDS is that it is only first order accurate on the basis of the Taylor Series Truncation Error (TSTE). It introduces a “false diffusion” when the flow and the grid lines are not aligned, increasing the robustness of the simulation, but producing erroneous results [1].

2.3.2 Second order UDS

To derive an expression for the approximated face value of the flow variable ϕ at the face w for the flow conditions in figure 2.1 using the second order UDS, it is assumed that the spatial derivative of the flow variable

ϕ is constant between nodes w and WW . After a minor derivation, it is found that

$$\phi_w = \frac{3}{2}\phi_W - \frac{1}{2}\phi_{WW}.$$

A major drawback of the second order UDS is that it is only conditionally bounded. Coefficients of the discretized equation can become negative under certain flow conditions, which can give rise to stability concerns and unbounded solutions. The accuracy with respect to TSTE of the second order UDS is second order, a significant improvement from its lower order equivalent [6].

2.4 Cooling system

The operating pressure and temperature of internal combustion engines are high, placing high demands not only on the material of many of the components, but also on the cooling system. Moreover, even though today's car engines are considered to be energy efficient, the efficiency is just around 30 – 40 %. Hence, the cooling system has to be able to handle the waste heat from many areas of the engine to prevent overheating of sensitive components.

Furthermore, as the comfort of cars continues to increase, there is a need to keep the passenger compartment at a suitable temperature. These requirements has to be fulfilled even when the car is at standstill in hot conditions, which further complicates the task of dimensioning heat exchangers.

A simplified block scheme of a cooling system circuit can be seen in figure 2.2.

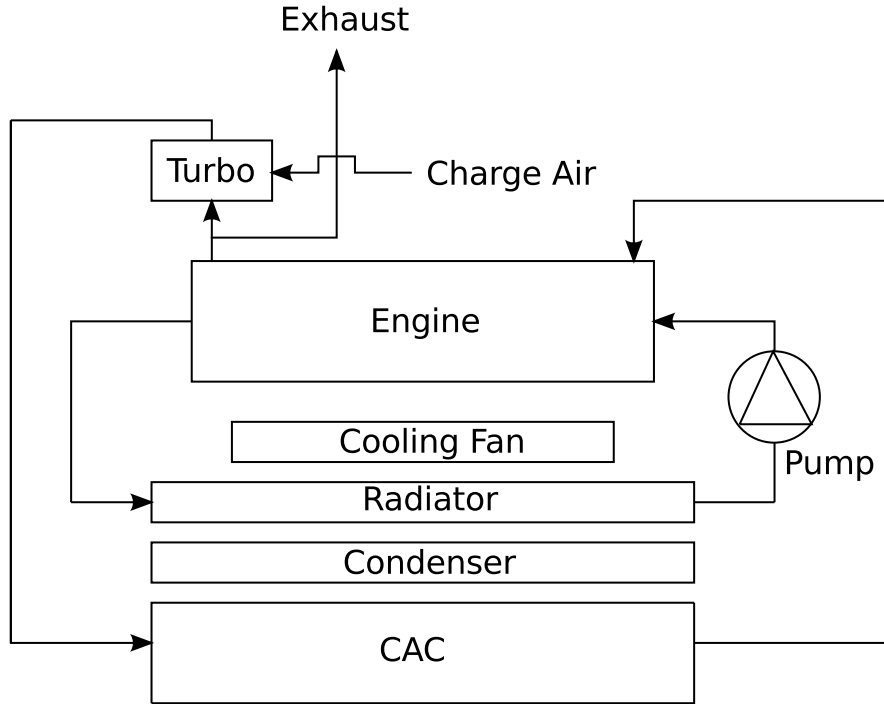


Figure 2.2: *Simplified cooling circuit, excluding the AC part.*

2.4.1 Radiator

To avoid irreparable damage on the engine, the engine block and cylinder head are cooled continuously. The system is water based, using a mixture of water and glycol to transport the waste heat. A common design of the system is a combined circuit through the engine components, transmission, and the radiator, but it differs between manufacturers and models. An additional radiator circuit is for example sometimes used, parallel to the main circuit, to cool the transmission oil.

The principle is straightforward; the coolant travels through the circuit from the engine where it is heated, to the radiator where it dissipates heat to the ambient air. The radiator core is designed in such a way that the surface to volume relation is high, transporting the coolant in narrow passages over which the air can flow.

2.4.2 Charge air cooler

Most modern engines are fitted with a turbocharger that compress the intake air in order to increase the amount of air in the cylinder. This is desirable, since it allows a higher power output while keeping the fuel consumption at a minimum.

When the air is compressed, its temperature increases. Too high charge air temperatures is unwanted, since it may lead to premature ignition of the fuel as well as increased engine temperatures. A remedy is to install a Charge Air Cooler (CAC), which cools the air before it is fed into the cylinder. The most common approach is to lead the compressed charge air through a heat exchanger in the front of the car, but a water based system in a fashion similar to the radiator circuit can also be used.

If the CAC is well designed, meaning that the pressure drop is low compared to the temperature decrease, the charge air density will increase. This allows for an even higher air mass flow into the engine, which will increase the performance of the turbocharger.

2.4.3 Condenser

One major challenge when dimensioning the heat exchangers is, as mentioned above, to keep the temperature in the interior of the car at a comfortable level. The condenser is responsible for this heat exchange between the refrigerant in the AC system and the ambient air. Since the condenser operates at lower temperatures than the radiator, it is always mounted in front of the radiator.

2.4.4 Axial fan

To increase the air mass flow through the heat exchangers a fan, in some cases more than one, is often mounted behind the cooling package of the car. Even at medium vehicle speeds, the impact of the fan is of great importance to the overall cooling performance, while it is of uttermost importance in order to handle the cooling demand at lower speeds.

2.5 Fan modeling

Turbomachinery is present in the underhood compartment in a variety of different applications, the cooling fan being the most important in the current context. It can influence the performance of the cooling system considerably, especially at low vehicle speeds.

It is of uttermost importance to model the fan in an appropriate way to be able to predict its effect on the flow, both through the heat exchangers and the rest of the underhood compartment. Several modeling approaches are implemented in commercial softwares at present, with varying levels of accuracy and demand of computational resources. The most important of those include the momentum source method, the Multiple Reference Frames (MRF) approach and simulations using a sliding mesh interface. The momentum source method does not use any geometrical representation of the fan, but modifies the source term in the Navier-Stokes equations to account for the influence of the fan. This requires substantial amount of experimental data to be able to predict flow quantities in an appropriate manner, limiting the design possibilities in the vicinity of the fan. The sliding mesh methodology is a transient model, which severely increases the computational demand. Hence, in the current context, the MRF model seems to be the method of choice since it offers a good balance between accuracy and computational expense.

2.5.1 Single and multiple reference frames

By using a rotating frame of reference, the rotation of a geometry can be taken into account by a transformation of the Navier-Stokes equations. This enables a steady state simulation of the moving part using a stationary mesh. Figure 2.3 illustrates the change of coordinate system, where the dotted line can be either the domain boundary or an interface, resulting in single and multiple reference frames respectively.

The Navier-Stokes equations used in the MRF approach is readily derived by considering the time derivative of the position vector in the rotating frame of reference

$$\left(\frac{dr_i}{dt}\right)_I = \left(\frac{dr_i}{dt}\right)_R + \varepsilon_{ijk}\Omega_j r_k. \quad (2.25)$$

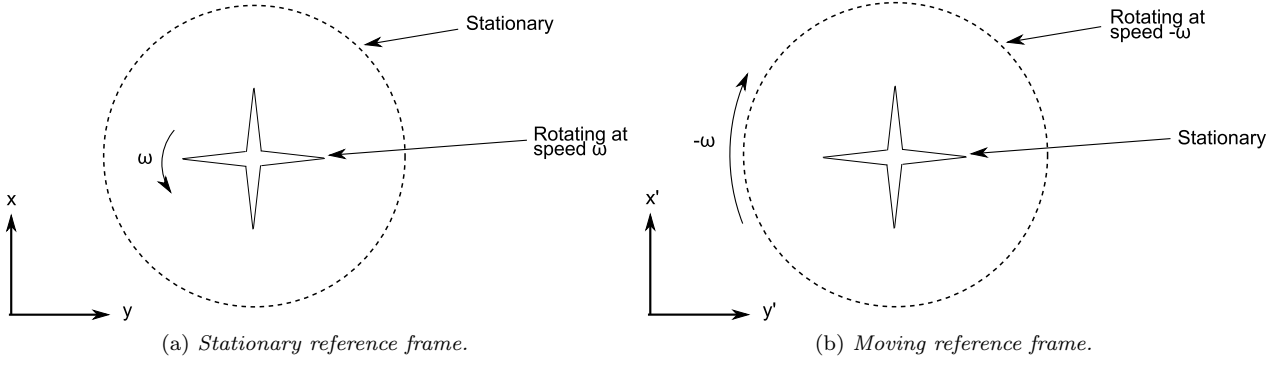


Figure 2.3: Comparison of the same physical domain using two different reference frames.

The indices I and R denote the inertial and rotating frame of reference respectively while Ω_j is the rotational vector. The time derivative can be seen to consist of two parts, one representing the change of the position vector itself and one representing the rotation of the frame. Using the same procedure with r_i exchanged for the inertial velocity $u_{I,i} = u_{R,i} + \varepsilon_{ijk}\Omega_j r_k$ yields an expression for the acceleration according to

$$\left(\frac{du_{I,i}}{dt}\right)_I = \left(\frac{du_{R,i}}{dt}\right)_R + \varepsilon_{ijk}\frac{d\Omega_j}{dt}r_k + 2\varepsilon_{ijk}\Omega_j u_{R,k} + \varepsilon_{ijk}\Omega_j \varepsilon_{klm}\Omega_l r_m. \quad (2.26)$$

The terms $2\varepsilon_{ijk}\Omega_j u_{R,k}$ and $\varepsilon_{ijk}\Omega_j \varepsilon_{klm}\Omega_l r_m$ represent the Coriolis effect and centrifugal acceleration, respectively. Combining (2.26) with the Navier-Stokes equations, excluding the time dependent terms, gives the steady system of equations

$$\frac{\partial u_{R,i}}{\partial x_i} = 0, \quad (2.27)$$

$$\frac{\partial}{\partial x_j}(u_{R,i}u_{R,j}) + 2\varepsilon_{ijk}\Omega_j u_{R,k} + \varepsilon_{ijk}\Omega_j \varepsilon_{klm}\Omega_l r_m = -\frac{\partial}{\partial x_i}\left(\frac{p}{\rho}\right) + \nu \frac{\partial^2 u_{R,i}}{\partial x_j \partial x_j}. \quad (2.28)$$

As an example the open source solver **simpleSRFFoam**, part of the **OpenFOAM** distribution, solves the equation system presented above. In applications where more than one reference frame is needed, i.e. for which the MRF method is more suitable, it is convenient to further develop the left hand side of (2.28) to solve for the velocity in the inertial frame of reference. This can be achieved by exchanging one of the relative velocities in the term $\frac{\partial}{\partial x_j}(u_{R,i}u_{R,j})$ by $u_{I,i} - \varepsilon_{ijk}\Omega_j r_k$, resulting in

$$\frac{\partial u_{I,i}}{\partial x_i} = 0, \quad (2.29)$$

$$\frac{\partial}{\partial x_j}(u_{R,i}u_{I,j}) + \varepsilon_{ijk}\Omega_j u_{I,k} = -\frac{\partial}{\partial x_i}\left(\frac{p}{\rho}\right) + \nu \frac{\partial^2 u_{I,i}}{\partial x_j \partial x_j}. \quad (2.30)$$

$$(2.31)$$

The final expressions give a system of equations which can be solved in the entire domain given a region where $\Omega \neq 0$, outside which the system of equations reduce to the usual Navier-Stokes equations.

Hence, a solver has to implement several things, e.g. adding $\varepsilon_{ijk}\Omega_j u_{I,k}$ as a source term and transform the given absolute velocity to a relative in the specified MRF zones, in order to be able to handle a MRF model of rotating machinery. Special attention should be paid to the included zone boundaries, defining which should rotate with the reference frame and which should act as stationary walls.

Limitations

Many studies concerning the limitations and consistency of the MRF model can be found in the literature. It is found that the choice of rotating region and frozen fan position have a large impact on the accuracy of the model, as described by Wang et al. [7] and Gullberg [8].

The authors state that in order for the MRF method to be valid, the inlet and outlet interfaces must be placed in regions where the steady state velocity and pressure profiles are axisymmetric across the interface. In most cases, e.g. in the cramped space in the engine bay, the choice of MRF region is highly limited by the surrounding, nonsymmetrical geometry, complicating the task of defining the rotating region.

The general trend in [8] indicates that the influence of the frozen fan position becomes significant when the number of blades of the fan reduces, resulting in outflow profiles dependent on the angle, and when the blockage is asymmetrical.

The CFD engineer should be aware that applying the MRF method causes more or less erroneous results and effort has to be taken to obtain valid results.

2.6 Heat exchanger modeling

Three different types of heat exchangers are often present in the underhood compartment, namely a radiator, a charge air cooler and a condenser. As described in section 2.4, their design with narrow passages to increase the surface to volume ratio complicates the task of resolving the flow around the real geometry. In a RANS simulation, one can rarely afford to resolve geometrical detail of such small scales.

A common approach when modeling a heat exchanger is to replace the input geometry with an empty cuboid and model this as a porous region. This takes into account the pressure drop and flow resistance across the heat exchanger core, while an additional model has to be applied to account for the heat transfer. This section will briefly introduce methods available in STAR-CCM+ to model heat exchangers.

2.6.1 Porous media model

Since it is not the details of the flow inside the heat exchanger that is of interest in many CFD simulations focused on an underhood thermal analysis, modeling the heat exchangers as a porous medium is a suitable method. The macroscopic effects of the heat exchanger geometry on the flow can be taken into account by introducing a source term in the momentum equation, according to [9]

$$s_{p,i} = -P_{ij}^V u_j + P_{ij}^I |u| u_j, \quad (2.32)$$

where P_{ij}^V and P_{ij}^I is the viscous and inertial porous resistance tensors. The effects of the porous region is hence taken into account similarly to Darcy's law, but with an additional non-linear term. Care has to be taken when defining the coordinate system of the porous region. In many applications the geometry of the porous material favors flow in a specific direction, for example the heat exchangers considered in the current thesis which have pores mainly oriented in the front-to-back-direction of the car. The usual approach is then to define the resistance in the favored direction to satisfy experimental results and then increase the resistance in the other directions by two to three orders of magnitude.

2.6.2 Single stream heat exchanger model

Instead of solving the heat transfer from one continuum to another, this model introduces an enthalpy source to account for a prescribed heat flow in a specified region. Only one of the streams are modeled explicitly, while the other is assumed to have a uniform temperature.

The main difference between using this option and just prescribing a total volumetric heat source is that the single stream approach takes the local mass flow into account, whereas the total volumetric heat source does not. This ensures that more heat is exchanged in areas where the mass flow is high, thus avoiding problems that can arise in regions where the mass flow is close to zero when using the total heat source model. It also includes a number of different parameters to better control the model, e.g. a minimum temperature difference parameter able to limit the temperature of the fluid stream in order for it not to increase (or decrease) nonphysically close to the uniform temperature of the heat exchanger.

In STAR-CCM+ the local heat transfer of a cell i is computed as [9]

$$Q_i = Q_{tot} \frac{V_i u_i (T_{ref} - T_i)}{\sum_i V_i u_i (T_{ref} - T_i)}, \quad (2.33)$$

where Q_{tot} and T_{ref} are the prescribed total heat source and temperature of the heat exchanger, while V_i , u_i and T_i are the volume, velocity and temperature of cell i , respectively.

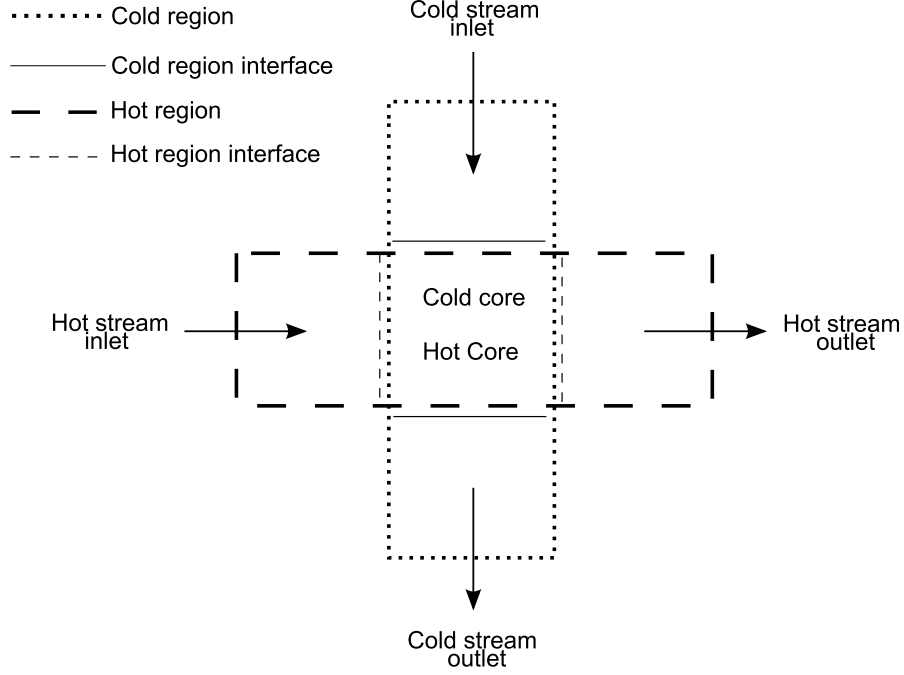


Figure 2.4: *Different regions and interfaces needed in the dual stream heat exchanger model.*

2.6.3 Dual stream heat exchanger model

As opposed to the single stream heat exchanger approach, the dual stream model couples the heat transfer between the hot and cold continua. In order for this model to achieve valid results, the cores of the different continua should consist of identical meshes. One should also strive to create conformal interfaces between the different regions of the same continuum, i.e. between the core regions and their respective in- and outlet region, as depicted in figure 2.4.

The dual stream heat exchanger approach of STAR-CCM+ calculates the local heat exchange using a specified or computed cell heat transfer coefficient [9]

$$Q_i = \pm (UA_l)_i (T_{hot,i} - T_{cold,i}), \quad (2.34)$$

where the sign of the term depends on whether it is the warm or cold continuum that is considered. A number of different methods are available to map experimental results to the simulation to obtain a correct heat transfer coefficient, UA_l . The “Q map” option is one example of available methods, important in the current context. It calculates the local UA_l number according to [9]

$$UA_l = \frac{\Gamma}{\Delta T_{net}}, \quad (2.35)$$

where ΔT_{net} is the net temperature difference defined as the summation of the temperature difference between the hot and cold stream times the volume of each cell in the heat exchanger core, divided by the average volume of a cell. Γ is calculated from the supplied Q map as

$$\Gamma = \frac{Q (\bar{T}_{in}^{hot} - \bar{T}_{in}^{cold})}{T_{user}^{hot} - T_{user}^{cold}}. \quad (2.36)$$

The temperatures in the denominator are the inlet temperatures of the both streams for which the Q map-defining tests are performed, while the temperatures in the numerator are mean temperatures on the inlet interface of the two regions in the simulation. The Q map is a table consisting of heat rejection values for different coolant and ram air mass flows, obtained from supplier CAE or testing, from which the value of Q is calculated by interpolation from the mass flows in the simulation.

2.7 Geometry representation

When information of a product is conveyed through an organization it is convenient to have a file format able to handle more than one representation of the geometry. This serves the purpose of minimizing the translations needed between different softwares used in the Product Lifecycle Management (PLM) system. A consequence which is not only practical, but also means that problems which can occur in the translation process resulting in dirty geometry can be avoided.

jt is a file extension used in the industry to communicate design information through an organization for design and visualization purposes. It is commonly referred to as a lightweight visualization format and has the capability to handle all major 3D Computer Aided Design (CAD) formats occurring in the different softwares used in the PLM system. Computer Aided Engineering (CAE) softwares, such as ANSA, are then able to convert the data contained in the **jt**-file to a desired representation of the 3D geometries. Two different representations will be briefly described below.

Triangulation representation of a surface

A triangulation of a surface is an approximate, facet-based description of the input geometry. It ignores many of the entities of a complete CAD model, e.g. curves, lines and points, and only stores a list of facet data. This data consists of three vertices, which together forms a triangle.

stl is a format using this representation of the geometry. Apart from the vertices it also contains the orientation of the facets which are grouped together to form different patches of the geometry. An ASCII definition of a patch and its associated two facets in a **stl** file can be seen in listing 2.1.

```
solid patch1
  facet normal 0 0 -1
    outer loop
      vertex 0.003234 0.00317894 -0.01375
      vertex 0.00232785 0.00310005 -0.01375
      vertex 0.002302 0.00389537 -0.01375
    endloop
  endfacet
  facet normal 0 0 -1
    outer loop
      vertex 0.002302 -0.00389537 -0.01375
      vertex 0.00232785 -0.00310005 -0.01375
      vertex 0.003234 -0.00317894 -0.01375
    endloop
  endfacet
endsolid patch1
```

Listing 2.1: Definition of a facet in a **stl** file.

NURBS representation of a surface

Non-Uniform Rational B-splines (NURBS) is a tool in geometric representation for design and data exchange purposes, recognized and used by many standards such as IGES and STEP. This is largely due to the fact that NURBS provide fast and stable algorithms for representing both analytical and arbitrary shapes [10]. NURBS display a few favorable characteristics. They:

- Are storage efficient (points, weights, start/end points and knot vector are stored)
- Are invariant under affine transformations, operations are done on the points only
- Have local support (a control point only affects the region where it is active)
- Introduce weights to allow for more exact representations without increasing the number of points.
- Define a surface as the tensor product between two curves.

A NURBS curve of degree p is commonly defined as [10]

$$\mathcal{C}(u) = \sum_{i=0}^n R_{i,p} P_i,$$

where $(P_i)_{i=0\dots n}$ are the control points (n to the number) and $R_{i,p}$ is given by

$$R_{i,p}(u) = \frac{N_{i,p}(u)w_i}{\sum_{j=0}^n N_{j,p}(u)w_j}.$$

w_i are the weights and the term $N_{i,p}(u)$ refers to the p -th degree B-spline basis function, non-zero in a specific knot interval and defined as

$$\begin{aligned} N_{i,0}(x) &= \begin{cases} 1, & \text{if } t_k \leq x < t_{k+1} \\ 0, & \text{else} \end{cases} \\ N_{i,p}(x) &= \frac{x-t_i}{t_{i+k}-t_i} N_{i,p-1}(x) + \frac{t_{i+k+1}-x}{t_{i+k+1}-t_{i+1}} N_{i+1,p-1}(x), \end{aligned}$$

where t_i are the knots defining where the basis functions are active.

2.7.1 Geometry clean-up

Non-manifold and intersecting surfaces, disconnected edges, gaps and excessive details are abundant in the typical input geometry in an underhood CFD analysis. Such a geometry representation is unsuitable for mesh generation and preparing it has been, and remains to be, one of the most time consuming tasks for the CFD engineer. Geometry clean-up methods, such as surface wrapping algorithms, have therefore become important to reduce this both costly and repetitive task.

The general wrapping procedure can be summarized in three steps, visualized in figure 2.5.

Surface wrap algorithm

The surface wrap algorithm will be briefly explained in this section; its general procedures should be similar to the algorithms applied by commercial softwares used in the thesis work. However, the exact solution order, optimization and search algorithms are trade secrets that differ between the softwares and companies.

In order for the shrink-wrap algorithm to be used, an initial surface needs to be generated. A convenient way to do this is to generate an octree template mesh and identify which cells are intersected by the original geometry, as shown in figure 2.6 to 2.7b on pages 13–14. The triangulated surface can then be created using suitable surfaces of those cells. Depending on the prescribed settings of the surface wrap operation, e.g. contact prevention, and minimum and target surface size, the cells are refined to achieve the desired resolution of the input geometry, as seen close to the corners in figure 2.7b.

The next step of the algorithm is to snap it to the geometry. Finding the closest point on the geometry and projecting it to the corresponding face is one easy way to do this, however more refined methods are often applied in commercial softwares, taking feature lines and quality of the resulting mesh into account. The time needed to find the closest point on the surface geometry is reduced due to the fact that the set of faces to be considered is limited by which of the surfaces are contained by the octree cell element in question.

The resulting surface mesh, figure 2.7e to 2.7f on page 14, is then optimized, to e.g. smooth it, retain some of the feature lines of the original geometry, and improve the quality of the mesh.

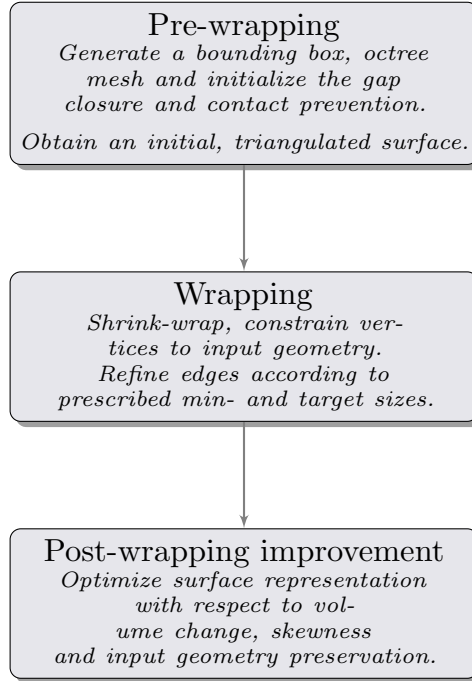


Figure 2.5: General procedure used in surface wrapping algorithms.

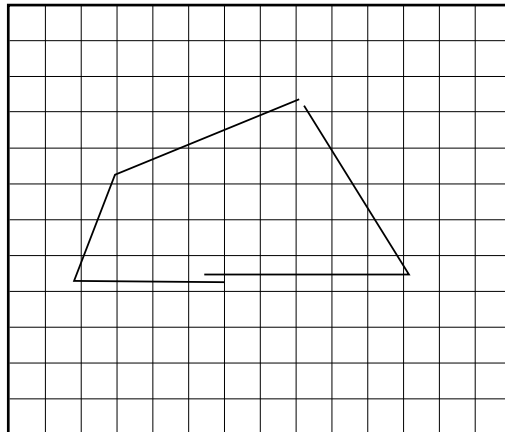
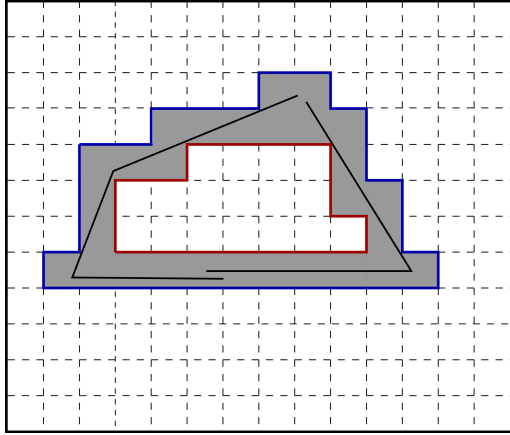


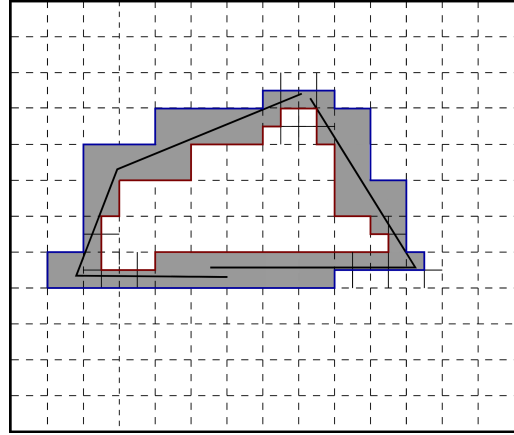
Figure 2.6: Initial octree mesh and incomplete, 2-dimensional CAD geometry.

- Intersected cells
 - Not intersected cells
— - Inner initial surface — - Outer initial surface

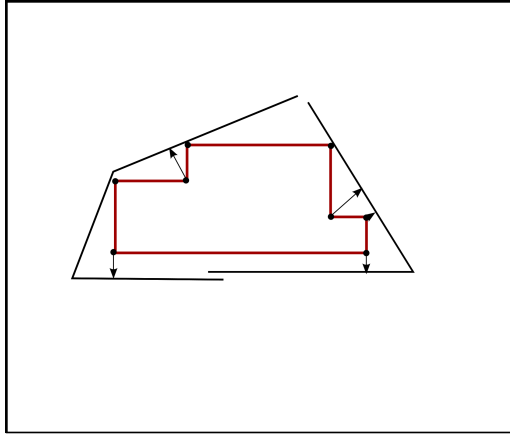


(a) Intersecting cells of the octree mesh and corresponding outer and inner initial surface.

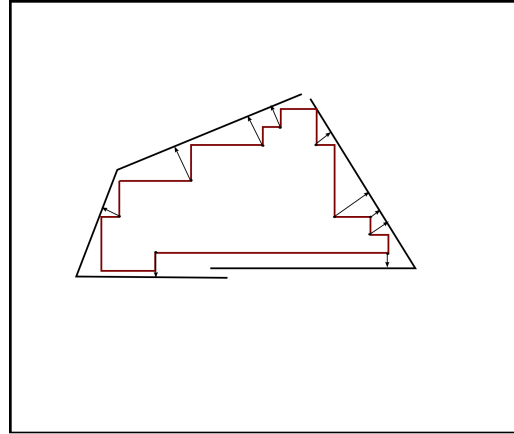
- Intersected cells
 - Not intersected cells
— - Inner initial surface — - Outer initial surface



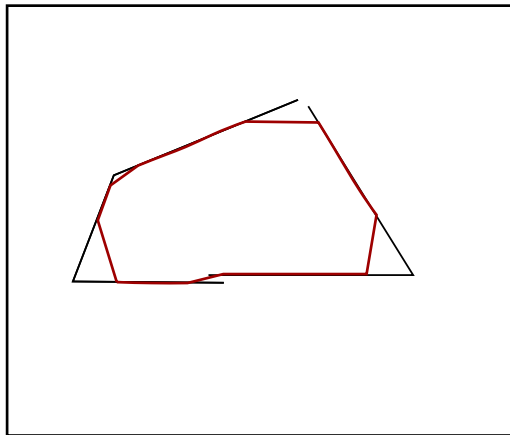
(b) Intersecting cells of the octree mesh and corresponding outer and inner initial surface, refined case.



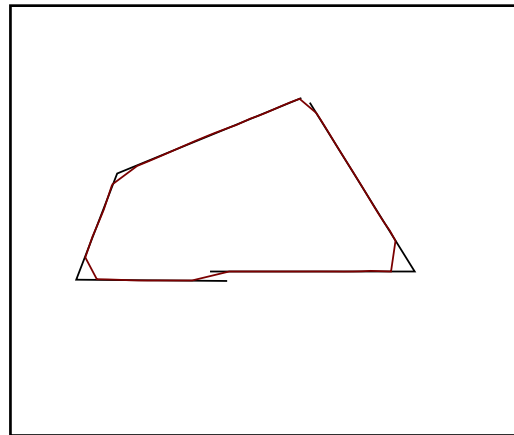
(c) Corner vertices being snapped to the initial surface, in this case the inner, to the input geometry.



(d) Corner vertices of the refined initial surface being snapped to the initial surface, in this case the inner, to the input geometry.



(e) Surface representation after projection, without any refinement.



(f) Surface representation after projection, with refinement.

Figure 2.7: The pre-wrapping and wrapping steps of the surface wrapper; without refinement to the left, with refinement to the right.

2.8 Volume meshing

This section will include a brief introduction of the utilities available to generate a volume mesh with the softwares used in the thesis work. The general methods for generation of hexahedral octree meshes and fully polyhedral meshes will be discussed in more detail.

A volume mesh used for CFD simulations can consist of several different cell types, of which the relevant ones are presented in figure 2.8. The tetra-, hexa-, and polyhedral types are named after the number of faces they consist of, with four faces for a tetrahedral, six faces for a hexahedral, and an arbitrary number of faces for a polyhedral cell. Prisms are a subclass of polyhedral cells, that are used to control the wall normal mesh resolution near walls.

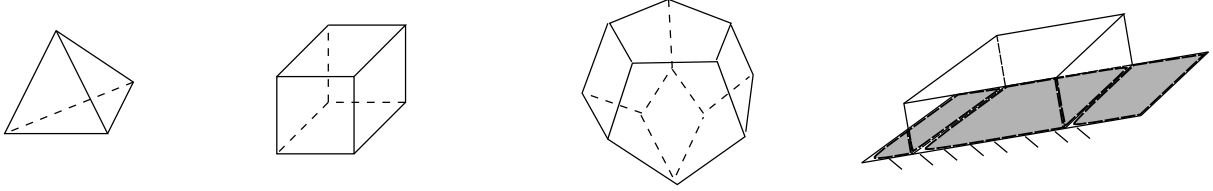


Figure 2.8: *Different cell types. From left to right: Tetrahedral, hexahedral, polyhedral, and prism.*

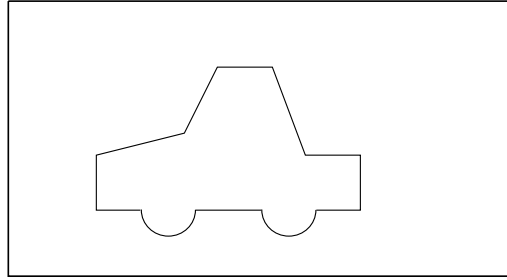
2.8.1 Hexahedral mesher

Similarly to the surface wrap algorithm the starting point of a hexahedral mesh generating algorithm is often a template mesh of user defined alignment and size, see figure 2.9a and 2.9b on the next page. The template mesh is then refined to take local surface- and volume sizes as well as growth rates into account, as seen in figure 2.9c.

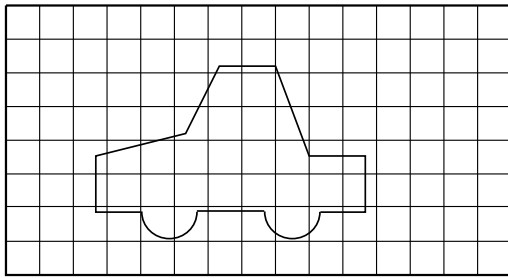
The process of cell removal is based on how many percent of each individual cell's volume lies within the geometry. In this step it is crucial to have a region enclosed by a bounding surface. Even when using the surface wrapper there is no guarantee that a closed representation will be achieved. To remedy this problem many existing commercial softwares have different options to ensure a closed surface representation. Given an enclosed surface, the resulting mesh is a coarse representation of the surface as illustrated in figure 2.9d.

The next step of the mesh generating algorithm is to snap the mesh onto the `stl`-surface to get a better representation of the underlying geometry. This step involves displacing vertices onto the `stl`-surface, e.g. by using a projection based algorithm, and displacing the internal mesh to take account for the displaced boundary and achieve a good quality mesh.

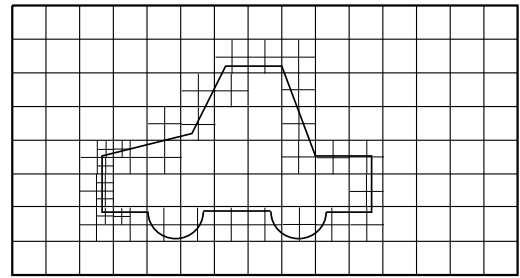
An additional step which is not illustrated in figure 2.9 is the ability to introduce prism layers. The general procedure of this utility is to displace the surface mesh, figure 2.9e, into the domain in the direction of the normal to the surface a distance corresponding to the prescribed thickness of the local prism layer. In complex geometries validation criteria of the perturbed mesh often require the thickness to be reduced in problematic regions. Mesh layers are then inserted and the two meshes are connected.



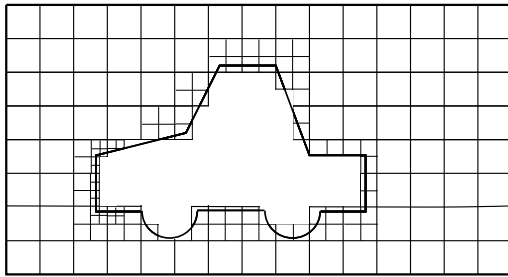
(a) Initial domain with 2-dimensional STL-surface.



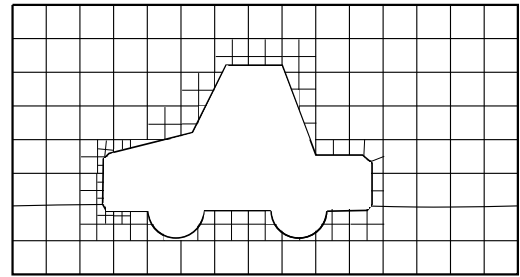
(b) Initial base size octree mesh.



(c) Refined base size mesh around STL edges and volume controls.

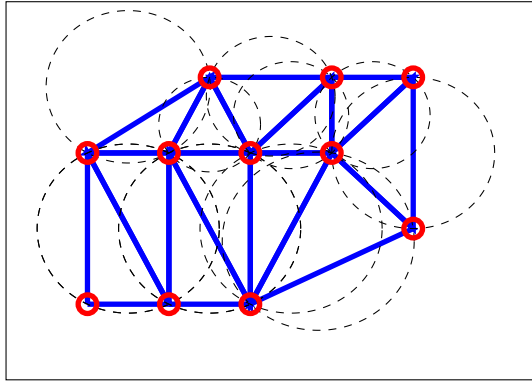


(d) Mesh after removing cells within the STL-geometry.

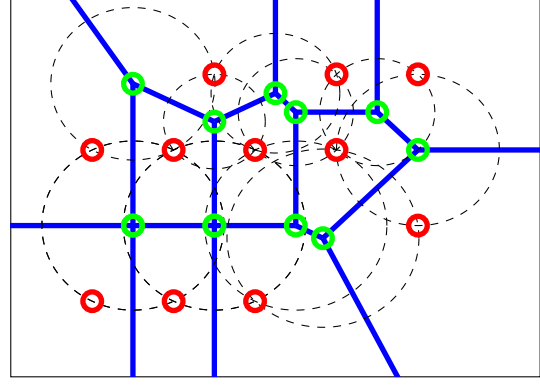


(e) Mesh after snapping it to the input geometry.

Figure 2.9: Visualization of an octree meshing tool. The steps may vary between different softwares, this procedure is inspired by the *OpenFOAM* utility *snappyHexMesh*.



(a) Delaunay triangulation of a set of 11 points in two dimensions, together with corresponding circumcircles.



(b) The Voronı diagram corresponding to the Delaunay triangulation in (a).

Figure 2.10: Delaunay triangulation and its dual Voronı diagram.

2.8.2 Polyhedral mesher

A polyhedral mesh in STAR-CCM+ is generated from an underlying tetrahedral mesh. The tetrahedral mesh is created using Delaunay triangulation, which is an algorithm that can be used to guide vertex creation. The procedure will be explained briefly below. The reader is referred to the literature for more information, for example the work by Frey and George [11]. Plenty of articles on the topic can be found, due to it being one of the major concerns in computational geometry.

Delaunay triangulation and the Voronı diagram

Given a set $\mathbb{S} \in \mathbb{R}^d$ of points $(p_i)_{i=1\dots n}$, the Delaunay triangulation is a triangulation such that each simplex has the empty circle (sphere in three dimensions) property as depicted in figure 2.10a. This means that the circumcircle C_i of every simplex entity of the Delaunay triangulation does not contain any point $p \in \mathbb{S}$.

Lemme général de Delaunay: Let \mathbb{T} be a given arbitrary triangulation of the convex hull of a set of points \mathbb{S} . If for each and every pair of adjacent simplexes in \mathbb{T} , the empty sphere criterion holds, then this criterion holds globally and \mathbb{T} is a Delaunay triangulation [11].

The Delaunay triangulation has several properties making it suitable for mesh generating and mesh refinement algorithms.

The Voronı diagram consists of a set of regions, R_i , defined as [11]

$$R_i = \{p : d(p, p_i) \leq d(p, p_j), \quad \forall j \neq i\}, \quad (2.37)$$

where $d(p, p_i)$ is the distance between an arbitrary point $p \in \mathbb{R}^d$ and $p_i \in \mathbb{S}$. Hence R_i is the region containing points p that is closer to p_i than any other point p_j in \mathbb{S} . The Voronı diagram is the dual of the Delaunay triangulation and can be readily achieved by connecting the centers of the circumcircles of the Delaunay simplexes as shown in figure 2.10b.

Polyhedral meshing

Given a tetrahedral mesh, which can be generated in several ways using Delaunay based methods, e.g. the incremental method described in [11], the polyhedral mesh can be constructed using a dualization scheme. In principle this could mean that the Voronı diagram, computed from the triangulation, and its resulting closed, convex and non-overlapping polygons can be used as cells in the polyhedral mesh.

3 Method

In this chapter, the methodology for the entire simulation process will be described. Two different methods were compared, one where the volume meshing was done using the software STAR-CCM+, and one where the meshing was done in Harpoon. For both methods, the simulations were performed in STAR-CCM+. The methods will be called *Method A* and *Method B* for the remainder of this report, where Method A refers to the method using STAR-CCM+ for meshing. A block scheme of the workflow of the two methods is presented in figure 3.1. When not explicitly stated otherwise, the described actions applies to both methods.

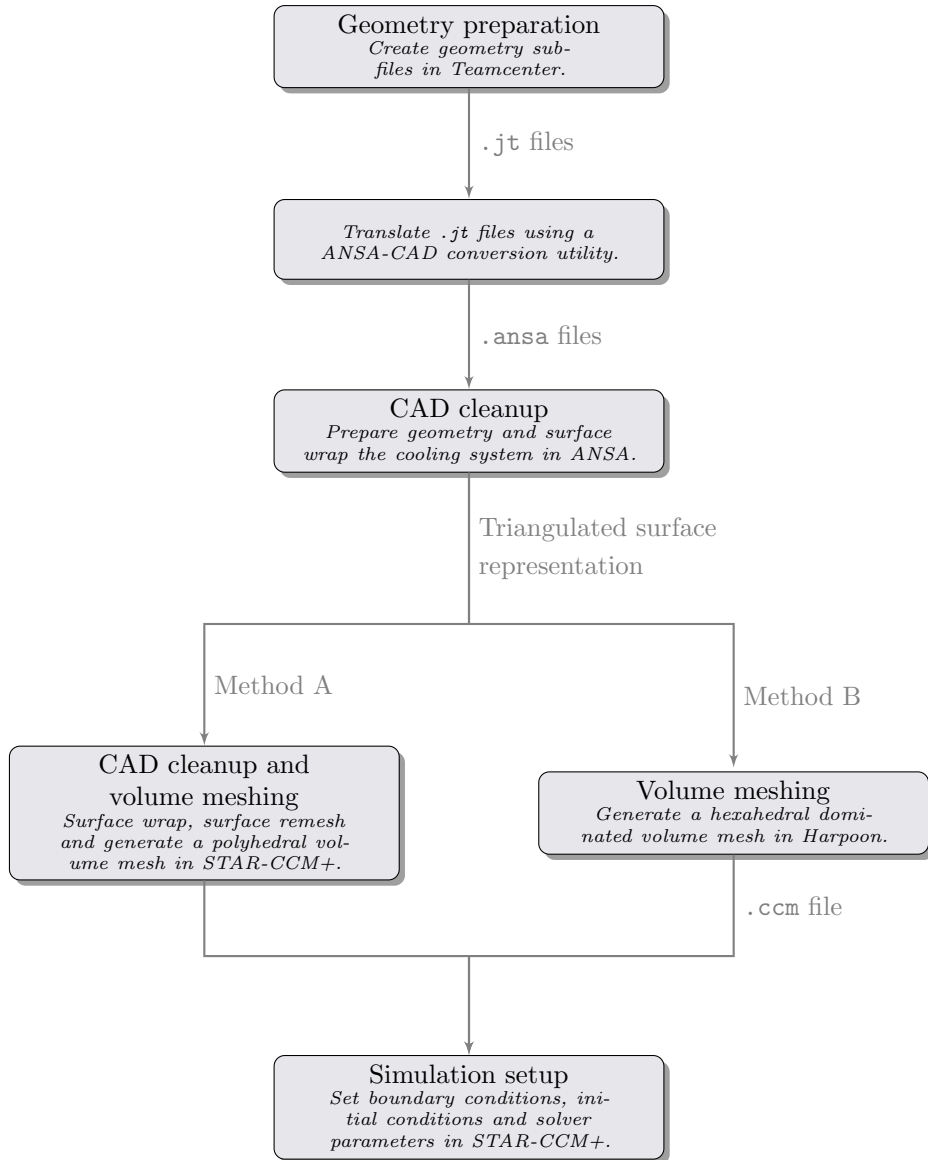


Figure 3.1: Schematic overview of the workflow for the two methods.

3.1 CFD software

A CFD simulation can roughly be divided into three parts; geometry preparation, meshing and simulation, each for which special software has been developed. This section will briefly describe the different softwares used in this thesis.

Teamcenter

When developing a complex product such as a car, a need arises to centralize all project data in a way that is easily accessible for the people involved in the process. To meet this need, Volvo uses the PLM (*Product Lifecycle Management*) system Teamcenter 9.1 from Siemens AG. This allows all engineers to access the latest design of a specific part, or even of a whole car project. The *jt*-file format described in section 2.7 is used to communicate up-to-date data of a model from Teamcenter to other softwares used in the PLM.

ANSA

As mentioned in section 3.3 it might happen that the geometries obtained from the PLM system are not suitable for meshing. In this project, ANSA 15.0 from BETA CAE Systems S.A. was used to clean up “dirty” input geometries, which involved both manual work through explicit geometry operations, as well as automated treatment through surface wrapping. The surface wrap algorithm in ANSA offers the user a great variety of settings to customize it to his needs. For example: the variable length wrapper offers the capability to capture most small scale details of the geometry, while the constant length wrapper offers a cheap and fast way of achieving an approximate representation of the geometry.

Harpoon

Harpoon is advertised by its creator Sharc Ltd as “*a fully automatic body-fitted hex-dominant mesher*” that uses the octree method to quickly generate high quality volume meshes. One of the software’s core strengths is its capability to generate volume meshes in a fast and effortless manner using batch scripts. The mesh generating utility is, however, in many areas more limited than its equivalent in other softwares.

STAR-CCM+

The maker of STAR-CCM+, CD-adapco, claims that their software is “*the world’s most comprehensive engineering simulation inside a single integrated package*”. It integrates basic CAD functionality and advanced meshing capabilities into the same package as the CFD solver, which is capable of handling various physics such as multiphase flows, porous media, conjugate heat transfer and solid stress. A major advantage available in many of the mesh generating utilities in STAR-CCM+ is the part-based meshing procedure. It enables the use of pipelined operations, significantly decreasing the work needed when for example changes are done in the input geometry or the parameters in a preceding operation are changed. The pipelined operations automatically identifies the operations effected by the change and updates them to take the change into account.

3.2 Geometry preparation

The goal of this thesis was to develop a method for obtaining a complete CFD model of the underhood compartment of a car. In order to do this, the geometry of the car had to be transformed into a suitable finite volume representation of the fluid volume. The first step in this process was to obtain a geometry representation in form of CAD data. Due to availability of test data from wind tunnel experiments, an XC60 D4 model year 2014 with 8 speed automatic gearbox was used for implementation and verification of the model.

For both methods, a CAD model of the complete car was extracted from Teamcenter. The extraction was done in smaller parts in order to get more manageable files; parts were grouped together according to their function, which meant that for example the cooling system formed one group while suspension and steering formed another. Hence, a separate *jt*-file was obtained for each group.

The *jt*-files were then converted to **ansa** format using the CAD Translator Utility in ANSA. For method A, the faceted representation was extracted since none of the file formats available in STAR-CCM+ was able to handle NURBS geometries, while the NURBS geometry was used for method B since all surfaces were to be meshed in ANSA.

3.3 CAD cleanup

The obtained **ansa**-files were then imported into ANSA for cleanup, with the cooling package as the main focus area. The cleanup served two purposes; obtaining a valid surface representation for meshing, as well as simplifying the later stages of the process. To achieve the latter, small surfaces and part were grouped together to form larger entities, as can be seen in figure 3.2. Note that each color corresponds to one part.

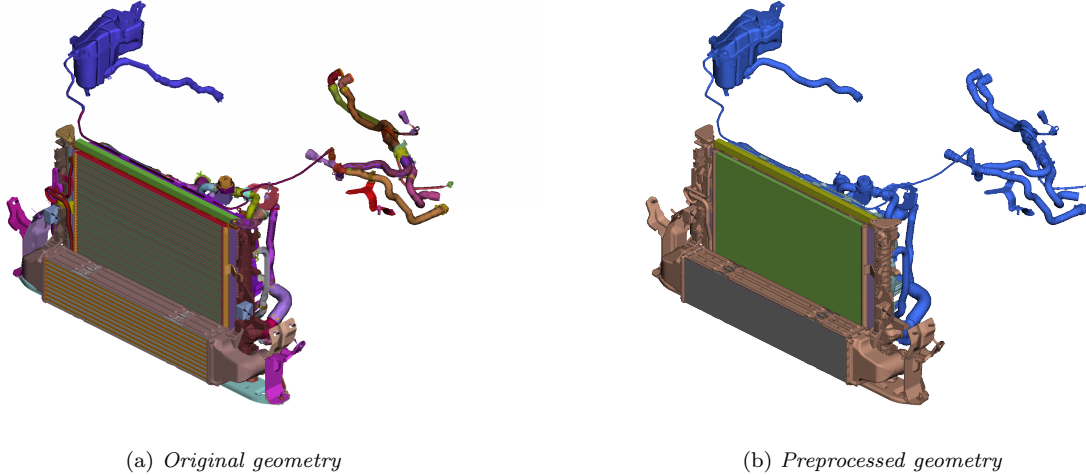


Figure 3.2: *Complete cooling package, before and after cleanup.*

Since the heat exchangers were to be modeled as porous media rather than resolving the complete flow field, the radiator geometries had to be replaced by cuboids. This involved a substantial amount of manual work, since they have to be of the correct dimensions, while still fitted to the surrounding geometries without any leaks. The result can be seen in figure 3.3 on the facing page, where the modeled inlet and outlet tanks on the radiator and CAC are visible. Instead of constructing small model tanks, the actual tanks could have been meshed and used, which is physically more correct, but would have resulted in a slightly higher computational cost.

The heat exchangers and model tanks were surface meshed using a basic STL algorithm¹, while the exterior of the actual tanks, tubes, pipes and attachments were surface wrapped using ANSA:s variable length wrapping function.

In order to use the MRF model, MRF zones had to be created in accordance with the outlines stated in section 2.5.1. However, due to the limited space around the fans, these zones could not be extended as far as desirable.

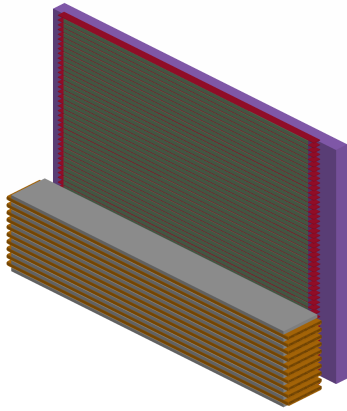
Like the creation of cuboids for the heat exchangers, the process of defining MRF zones required a considerable amount of manual time to be spent. This is due to the surfaces that has to be created, which can be seen as slightly transparent in figure 3.4b on the next page.

While the manually created limiting surfaces of the MRF zones were STL-meshed, the two impellers and the fan shroud were surface wrapped using the variable length wrapper. The resulting geometry including shroud, impellers and MRF zones can be seen in figure 3.4 on the facing page, while a more detailed view of the larger of the two impellers can be seen in figure 3.5. Note that the surface wrapping procedure removes several small gaps and bumps, as well as merging the roughly 100 surfaces to one; two consequences that are beneficial for the later stages of the process.

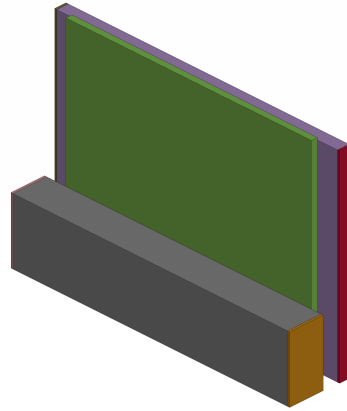
3.3.1 Method A

Since the whole car would have to be surface wrapped at a later stage for method A, the rest of the faceted geometry of the car was merely converted into a STAR-CCM+ compatible format and exported. The exception

¹The STL algorithm should not be confused with the **stl**-file format. The algorithm creates a surface triangulation without any restraints on the surface mesh quality, while the file format is a standard on how to store surface triangulations in files.

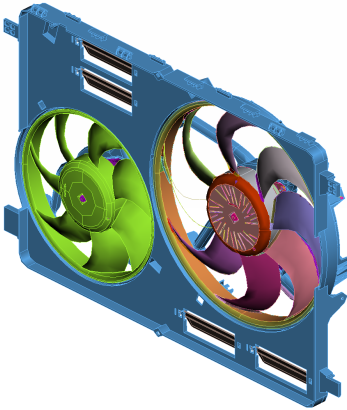


(a) *Original geometry*

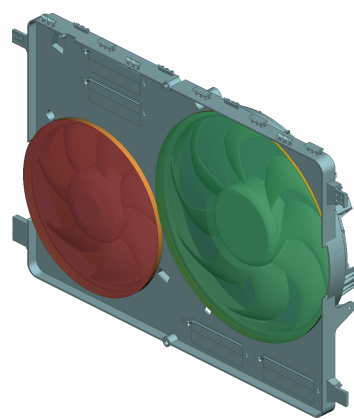


(b) *Preprocessed geometry with inlet and outlet tanks*

Figure 3.3: *Heat exchanger cores, before and after cleanup.*

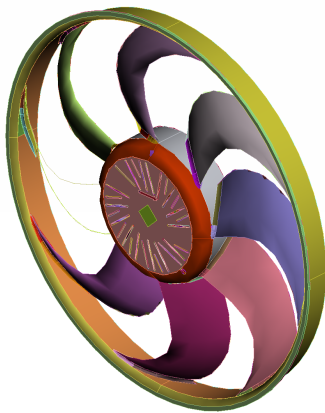


(a) *Original geometry*

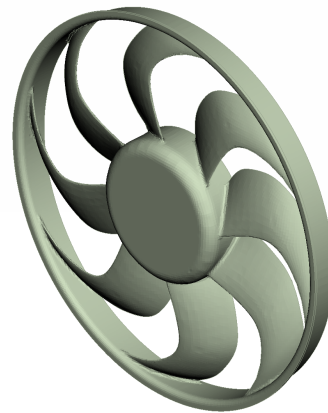


(b) *Preprocessed geometry with MRF zones*

Figure 3.4: *Fan with shroud, before and after cleanup.*



(a) *Original geometry*



(b) *Wrapped geometry*

Figure 3.5: *Impeller, before and after surface wrapping.*

was the car body, where some large holes due to missing parts were patched manually in ANSA before the triangulated surface was exported.

3.3.2 Method B

For method B, all parts except the cooling package were surface wrapped in ANSA. In order to cut down on the time required for preprocessing, the constant length wrapper was used whenever deemed possible. The “smooth” variant of the constant length method was used for all parts except the body, which meant that much of the geometry description was lost. Figure 3.6 displays the engine with gearbox, electric system and air induction system, where this loss of detail can be seen. Note that most small ribbons, gaps and other insignificant details have been discarded, which is desirable. However, the wrapping has produced a slightly larger volume, meaning that the fluid volume in the engine bay will be smaller than it should. This is not desirable since it might have a significant impact on the flow field.

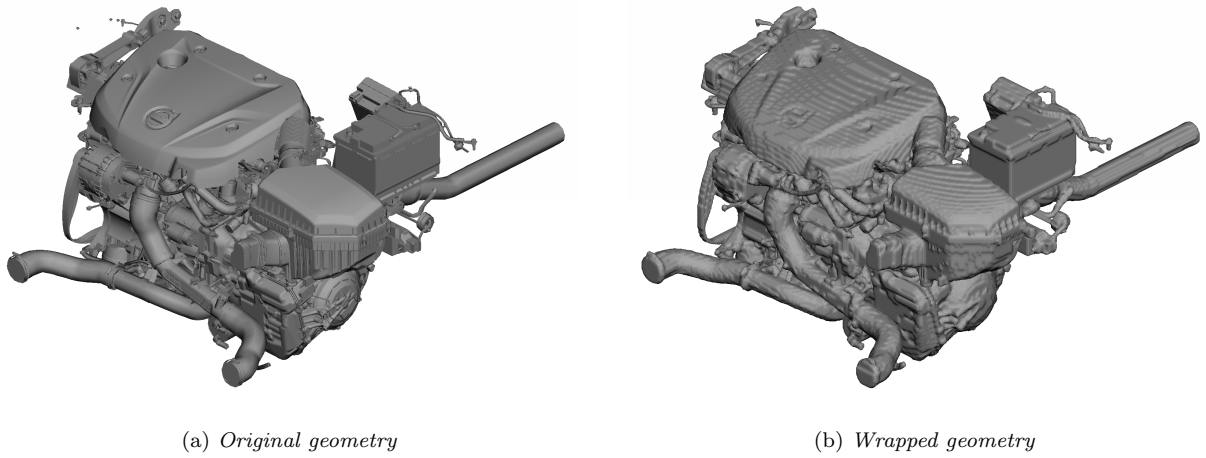


Figure 3.6: Engine with gearbox and air induction system. Battery included.

In order to simplify the meshing in Harpoon, the different surfaces were named in accordance with table 3.1. The first part of the names indicates whether the surface is a wall or is to be treated like an interface later in the process, while the trailing digit specifies the refinement level to be used on the surface. The middle part of the name identifies the location and name of the surface. For the sake of simplicity, the same names were used in method A.

All surfaces were finally exported to a single `msh`-file to be read into Harpoon. Since the `msh` file format is binary, it requires significantly less storage space compared to its alternative; an ASCII `stl`-file.

Table 3.1: Excerpt of surface names used in the model.

Surface	Description
wall-front-grill-6	Front grill
intercoolant-coolpack-cactank-in-6	Interface between CAC and its inlet tank
inter-mrf-large-out-7	Interface between backside of large MRF zone and surrounding air
rotating-wheel-front-right-5	Rotating front wheel

3.4 Volume meshing

The first step in creating the volume mesh was to decide the dimensions of the computational domain. For this project, a domain of size $23.0 \times 10.0 \times 5.42$ m, which can be seen in figure 3.7 on the next page, was used. Three refinement boxes were used; one semi-fine around the car, a fine under it and an even finer spanning the engine bay. Their effect on the final meshes can be seen in figures 3.8 to 3.9 on pages 25–26.

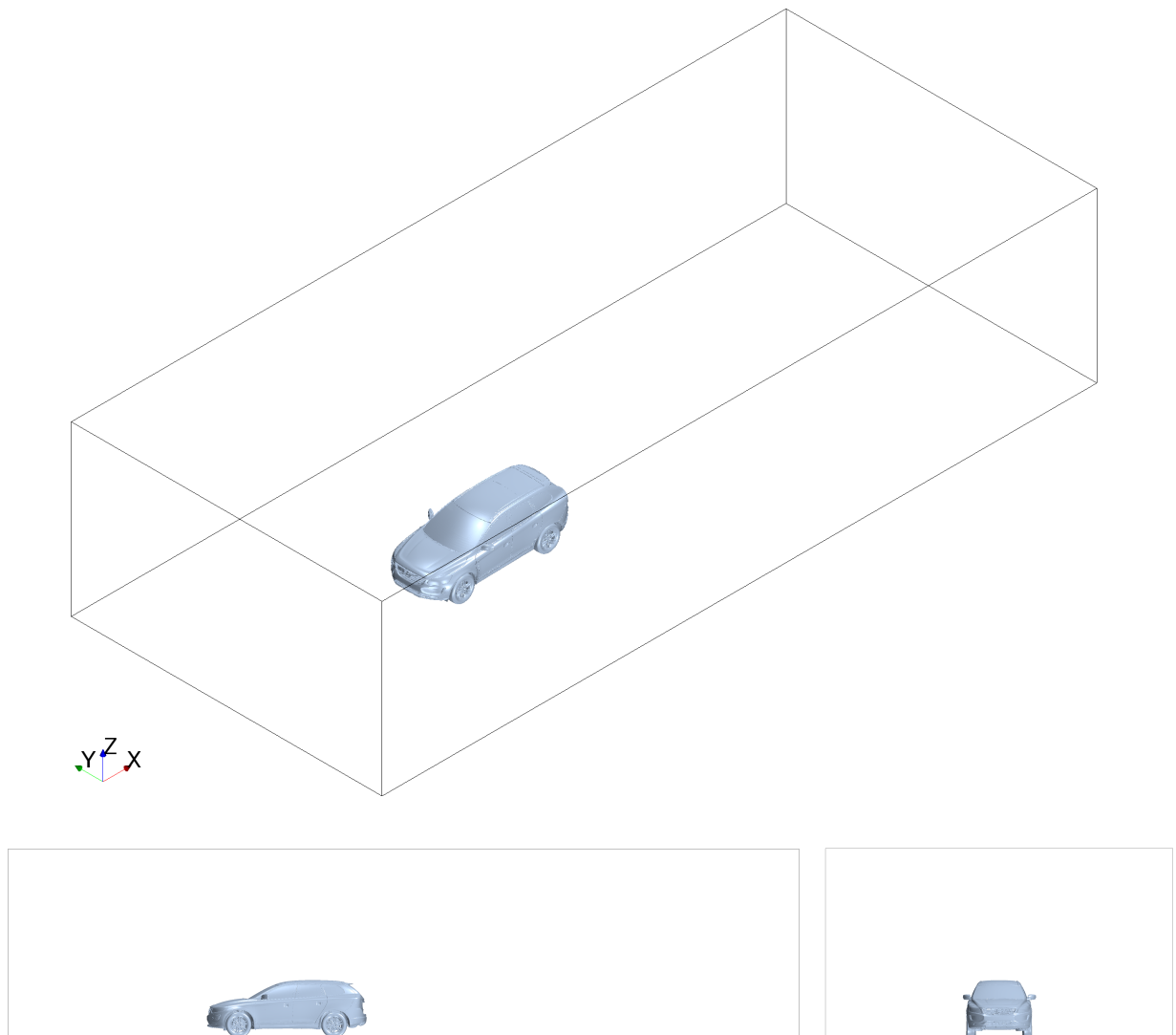


Figure 3.7: *The computational domain.*

Table 3.2: Regions in the model.

Region	Continuum	Description
air	air	Ambient air
rad-core	air	Cold side of radiator
cac-core	air	Cold side of CAC
cond	air	Condenser
large-fan	air	MRF zone for the large fan
small-fan	air	MRF zone for the small fan
rad-core-coolant	coolant	Hot side of radiator
radtank-in	coolant	Radiator inlet tank
radtank-out	coolant	Radiator outlet tank
cac-core-coolant	air-coolant	Hot side of CAC
cactank-in	air-coolant	CAC inlet tank
cactank-out	air-coolant	CAC outlet tank

In order to treat the different physics in the model, the mesh had to be split into several regions. Their names and a short description are found in table 3.2. Note that cac-core and cac-core-coolant occupy the same physical space, which also holds for rad-core and rad-core-coolant. This meant that cac-core-coolant and rad-core-coolant were constructed by copying the meshes in cac-core and rad-core during the case setup.

3.4.1 Method A

The process of obtaining a volume mesh in STAR-CCM+ from an imperfect geometry representation is divided into three major parts; surface wrapping, surface remeshing and volume meshing. Due to the so called *Parts based meshing* methodology in STAR-CCM+, these operations can be pipelined, which simplifies the meshing procedure, as mentioned in section 3.1.

Surface wrapping

One surface wrapping operation was created for each region in order to obtain a manifold surface representation. The wrap was refined in areas deemed important; mainly the engine bay, and especially around the cooling package. Parameters were chosen such that the cell count was minimized while still retaining important features.

Imprinting regions

In order to achieve conformal interfaces between regions, the interface surfaces in the adjoining regions had to be imprinted. An imprint operation between two regions will merge coincident faces so that the resulting surface is shared by the two regions, which is required if conformal interfaces are to be obtained. Imprint operations also create contact information between the regions, which will automatically create interfaces when the mesh is constructed, thus eliminating the need for manual creation of interfaces. Considering this, it would be beneficial to imprint all regions with each other in order to get conformal interfaces between all regions. Unfortunately, this was not possible in this simulation due to a possible bug in STAR-CCM+. Thus, only imprints between the heat exchanger cores and their tanks were made.

Surface remeshing and volume meshing

Surface remeshing and volume meshing are grouped together into an “Automated mesh operation” in STAR-CCM+, even though they are performed as separate operations when the meshing is executed. Furthermore, this kind of operation can also accommodate an “Automatic surface repair” operation, that is running between the surface remesher and the volume mesher. This feature was used in this project, since the surface remesher is prone to creating intersecting faces of dubious quality when the geometry is complex, which often is the case when analyzing underhood flow.

The polyhedral mesher, with the embedded prism layer mesher activated for all walls, was used to build the volume mesh. It generates a polyhedral mesh using an algorithm briefly described in section 2.8.

Parts created by the surface wrapper operations, i.e. closed manifold volumes, were then assigned to automated mesh operations. Since conformal interfaces between regions can be created only when the regions are meshed in the same automated mesh operation, it would be desirable to mesh all regions in one operation. However, due to the aforementioned imprint failure, each region was meshed separately except for the caccore, cactank-in and cactank-out as well as rad-core, radtank-in and radtank-out, which made two separate operations respectively. Hence, conformal interfaces were only created between the heat exchanger cores and their respective tanks and not between the overall domain and the heat exchanger and MRF regions in the air continuum. In such situations, the solver will force the mesh to be conformal at the interfaces by splitting the cells closest to the surface; an operation that might create bad or invalid cells.

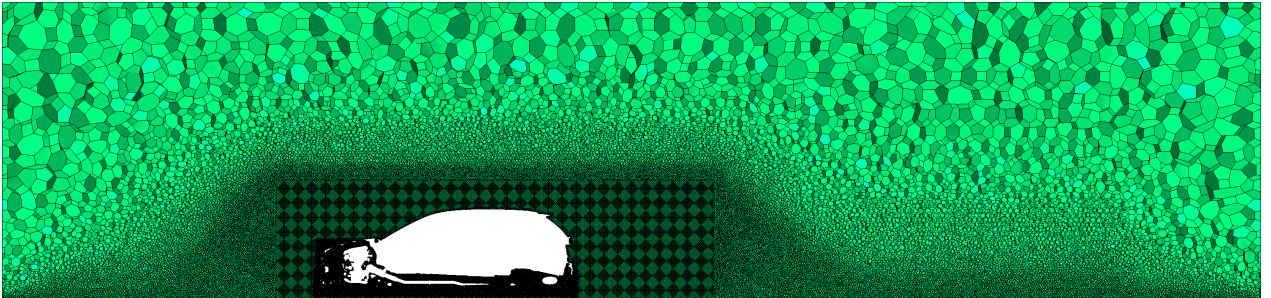
The final mesh, consisting of roughly 31 million cells, can be seen side by side with its Harpoon counterpart in figures 3.8 to 3.10 on pages 25–27.

3.4.2 Method B

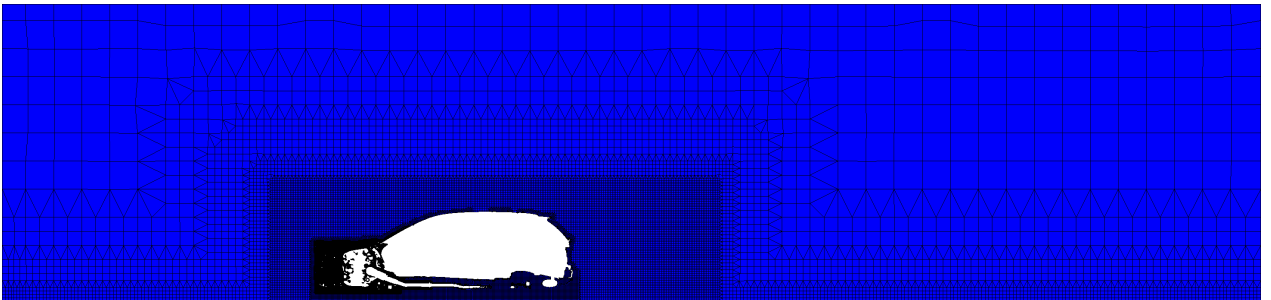
In order to streamline the meshing process, the volume meshing in Harpoon was done in batch mode, meaning that all setup was done by the script found in appendix A. The setup was simplified by the aforementioned naming conventions, which allowed refinement levels to be set using wildcards.

Contrary to STAR-CCM+, where one explicitly defines regions to mesh, Harpoon meshes all volumes inside the domain and deletes the ones it is not told to keep using the `vnamekeep` command. This has the advantage of always resulting in conformal interfaces between regions. However, small gaps in the surface might lead to meshing of volumes that are supposed not to be meshed. An example of this can be seen in figures 3.10a to 3.10d on page 27, where a comparison of the meshes generated by STAR-CCM+ in the left column and the Harpoon mesh in the right ditto shows that Harpoon are more prone to such leaks. This is especially clear for the beam in front of the radiators, and for the region just behind the engine bay. Such behavior is undesired since it not only increases the computational burden, but also might cause convergence issues.

Since Harpoon does not utilize the methods of expanding subsurfaces for creation of prism layers, but rather tries to build prism on the existing mesh, it ends up in dire straits when trying to build prisms on complex geometries. Thus, no prism layers were created, and the mesh had to be refined in all directions near walls in order to achieve a reasonable wall-normal resolution. This reflects in the fact that the finalized Harpoon mesh consists of about 73 million cells, more than twice as many as the polyhedral mesh created in STAR-CCM+. Considering figures 3.10e to 3.10h on page 27, depicting the meshed exterior of the car, this is clearly seen in that the Harpoon mesh is much denser, especially at the front of the car.

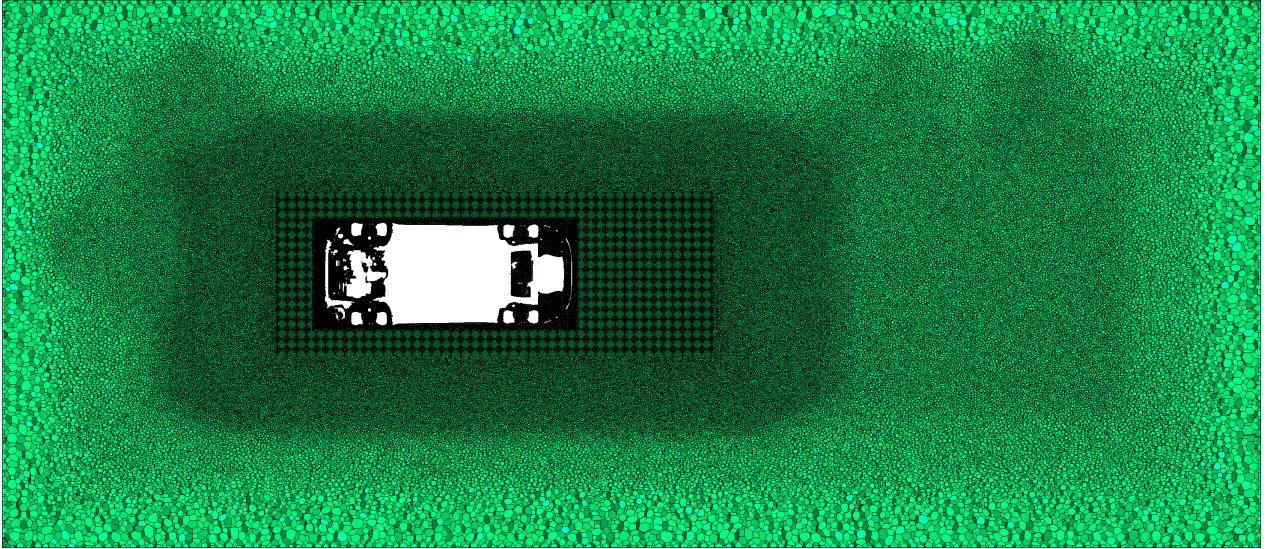


(a) Polyhedral mesh created in STAR-CCM+

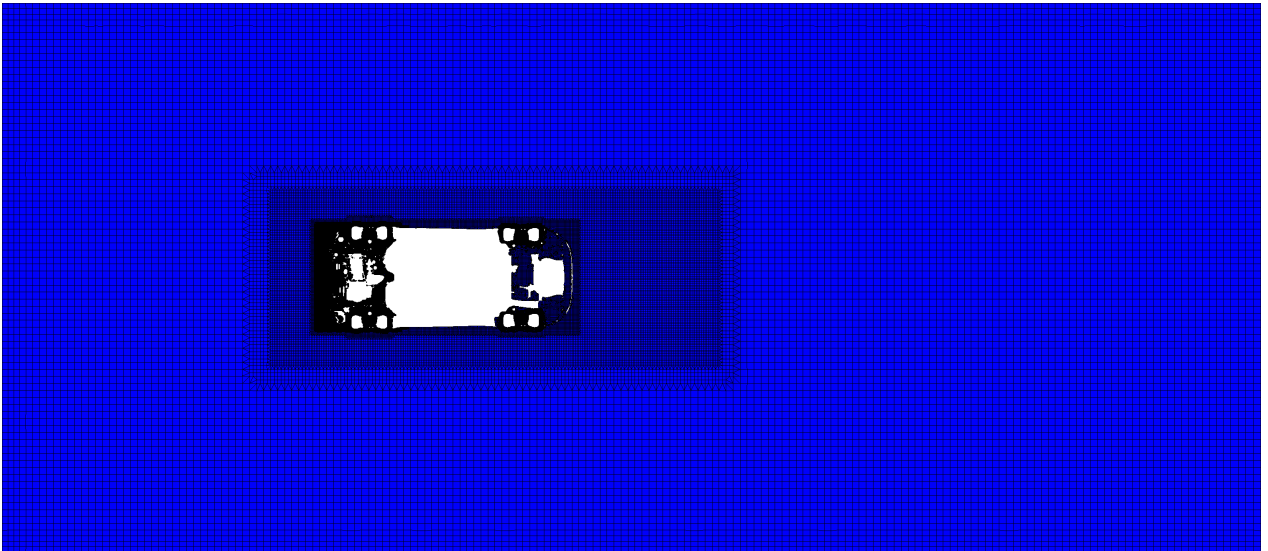


(b) Hexahedral mesh created in Harpoon

Figure 3.8: Side view at $y = 0$ m.



(a) *Polyhedral mesh created in STAR-CCM+*

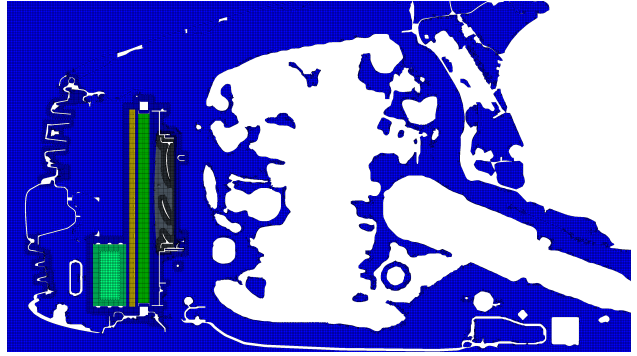


(b) *Hexahedral mesh created in Harpoon*

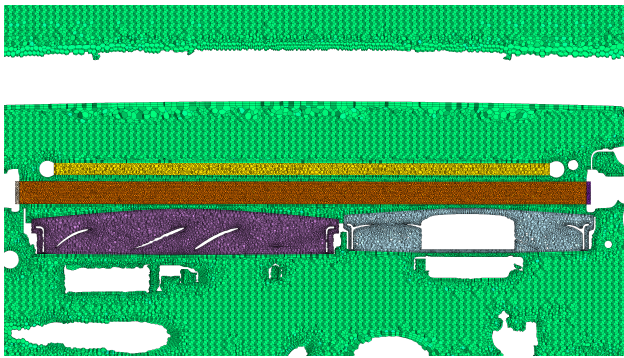
Figure 3.9: *Top view at $z = 0.6$ m.*



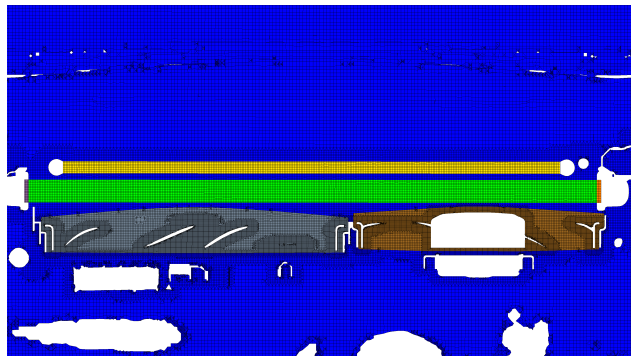
(a) *Method A: Engine bay at $y = 0$ m*



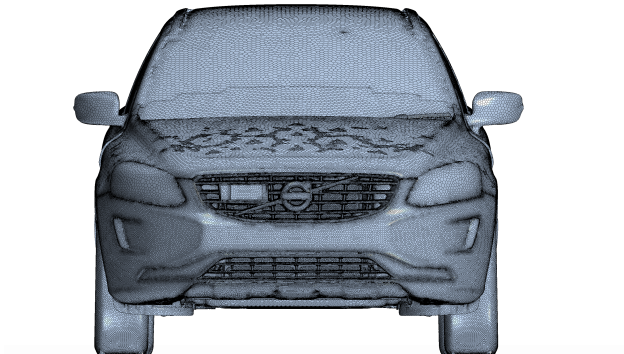
(b) *Method B: Engine bay at $y = 0$ m*



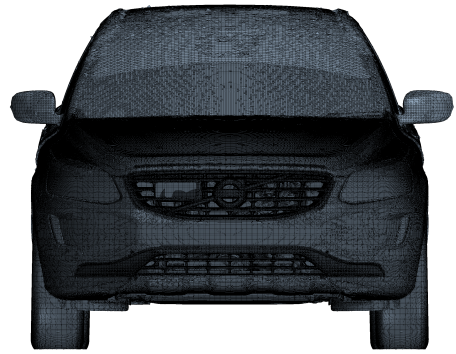
(c) *Method A: Cooling package at $z = 0.6$ m*



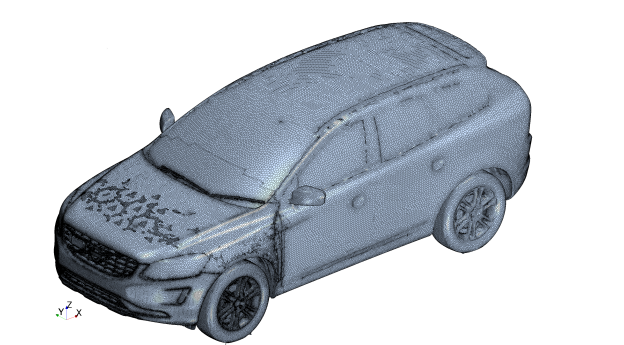
(d) *Method B: Cooling package at $z = 0.6$ m*



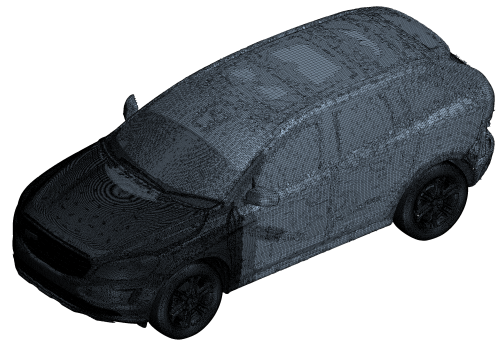
(e) *Method A: Front of car*



(f) *Method B: Front of car*



(g) *Method A: Exterior surface mesh*



(h) *Method B: Exterior surface mesh*

Figure 3.10: *Details of the meshes created with the two different methods.*

3.5 Case setup

As mentioned in section 3.4, the polyhedral mesh generating utility in STAR-CCM+ is capable of creating conformal mesh interfaces between different regions. When this utility is used successfully, no additional effort is needed in this stage of the setup. Importing the mesh in method B requires additional work however, to topologically connect the two regions and create the interfaces needed. This can be automated using a script, of which an example is presented in appendix C, or done manually in the GUI. There are relatively few differences between the methods and hence the following section will include a description of both methods simultaneously when possible. In case of a major difference, it will be highlighted in the beginning of each subsection.

3.5.1 Model selection

The major difference between the two methods when it comes to the case setup is that method A uses a coupled solver while method B applies a segregated approach. The coupled solver was chosen in the beginning of the project, due to several apparent advantages. One being the fact that it scales linearly with cell count, and another its capability to handle large source terms, e.g. rotating machinery, in a robust manner [9]. After extensive convergence issues, the segregated approach was recommended by the CD-adapco support to improve both convergence and accuracy of the simulation. Due to the late stage of the project, the segregated solver was not investigated thoroughly, but proved to help with some of the convergence issues of the simulation in method B.

The models selected for the different continua in method A and B can be seen in table 3.3 and 3.4 on the next page. The "Cell quality remediation" model identifies elements of poor quality in the mesh and modifies the way the gradients are computed in the vicinity of those cells, improving the robustness of the simulation while trying to minimize the effect on the accuracy [9]. It has proved an important feature to remedy convergence issues originating in bad quality cells.

3.5.2 Boundary conditions

Assigning boundary conditions to the different patches was done in the GUI. The conditions used in the validation case, presented in table 3.5 on the facing page, were set to consist with experimental data from wind tunnel tests of the HCTR1800 hill climb case. Note that the sides of the MRF-zones were explicitly set to be non-rotating in the original frame of reference. The walls of the porous regions have the condition of a slip wall, since they do not correspond to any real walls and no pressure loss should be introduced beyond that introduced by the porosity model. To ensure a well behaved flow in the internal regions of the dual stream heat exchangers, the outflow boundary conditions were set to be the negative of the corresponding inlet mass flow.

In a real car, the surface of the engine will transfer heat into the surrounding air through convection and radiation. These phenomena were not included here, since they were assumed to have a negligible impact on the flow through the heat exchangers.

3.5.3 Initial conditions

For method A, using the coupled solver approach, a grid-sequencing initialization is available. It generates an approximate inviscid solution for the flow variables of the problem. However, it was found that the solution it introduced was not always optimal and quantities were at times nonphysical in a high number of cells. Due to this unpredictable behavior, the common approach of defining starting values of the turbulent and flow variables was chosen, arriving at an equal approach for both the methods, summarized in table 3.6 on page 30.

Table 3.3: Selected models for method A.

Air	CAC-air	Coolant
Cell quality remediation	Cell quality remediation	Cell quality remediation
Coupled energy	Coupled energy	Polynomial density
Coupled flow	Coupled flow	Coupled energy
Gas	Gas	Coupled flow
Gradients	Gradients	Gradients
Ideal gas	Ideal gas	k-epsilon turbulence
k-epsilon turbulence	k-epsilon turbulence	Liquid
Realizable k-epsilon two-layer	Realizable k-epsilon two-layer	Realizable k-epsilon two-layer
Reynolds-Averaged Navier Stokes	Reynolds-Averaged Navier Stokes	Reynolds-Averaged Navier Stokes
Steady	Steady	Steady
Three dimensional	Three dimensional	Three dimensional
Turbulent	Turbulent	Turbulent
Two-layer all y+ wall treatment	Two-layer all y+ wall treatment	Two-layer all y+ wall treatment

Table 3.4: Selected models for method B.

Air	CAC-air	Coolant
Cell quality remediation	Cell quality remediation	Cell quality remediation
Gas	Gas	Polynomial density
Gradients	Gradients	Gradients
Ideal gas	Ideal gas	k-epsilon turbulence
k-epsilon turbulence	k-epsilon turbulence	Liquid
Realizable k-epsilon two-layer	Realizable k-epsilon two-layer	Realizable k-epsilon two-layer
Reynolds-Averaged Navier Stokes	Reynolds-Averaged Navier Stokes	Segregated flow
Segregated flow	Segregated flow	Segregated temperature
Segregated temperature	Segregated temperature	Reynolds-Averaged Navier Stokes
Steady	Steady	Steady
Three dimensional	Three dimensional	Three dimensional
Turbulent	Turbulent	Turbulent
Two-layer all y+ wall treatment	Two-layer all y+ wall treatment	Two-layer all y+ wall treatment

Table 3.5: Boundary conditions used in the simulations of the validation case.

Boundary	Boundary condition
Cac in	$\dot{m} = 0.099 \text{ kg s}^{-1}$, $T = 132.133 \text{ }^\circ\text{C}$
Cac internal walls	Slip wall
Cac out	$\dot{m}_{cac,out} = -\dot{m}_{cac,in}$
Inlet	$u_1 = 70 \text{ km h}^{-1}$, $T = 27 \text{ }^\circ\text{C}$
Outlet	Pressure outlet
Domain sides and roof	Symmetry
Domain floor	Prescribed velocity, $u_1 = 70 \text{ km h}^{-1}$
Front Wheels	Prescribed rotational velocity, 8.4 rps
MRF small and large sides	No-slip in the inertial frame of reference
Radiator in	$\dot{m} = 2.098 \text{ kg s}^{-1}$, $T = 105.256 \text{ }^\circ\text{C}$
Radiator internal walls	Slip wall
Radiator out	$\dot{m}_{rad,out} = -\dot{m}_{rad,in}$

Table 3.6: Initial conditions for the different continua. Reference pressure given inside parentheses.

Air continuum		Coolant continuum		CAC-air continuum	
Quantity	Value	Quantity	Value	Quantity	Value
u_i [km h ⁻¹]	[65 0 0]	u_i [m s ⁻¹]	[0 -0.15 0]	u_i [m s ⁻¹]	[0 -6.5 0]
T [°C]	27	T [°C]	105.256	T [°C]	132.133
p [Pa]	0 (0)	p [Pa]	0 (0)	p [Pa]	0 (268500)
ε [m ² s ⁻³]	1000	ε [m ² s ⁻³]	1000	ε [m ² s ⁻³]	1000
k [J kg ⁻¹]	0.001	k [J kg ⁻¹]	0.001	k [J kg ⁻¹]	0.001

Table 3.7: Values of the porous coefficients used in the simulations.

Region	P^i [kg m ⁻⁴]	P^v [kg m ⁻³ s ⁻¹]
Radiator external	[166.33, 1e4, 1e4]	[1e4, 870.01, 1e4]
Radiator internal	[100, 0, 100]	[100, 0, 100]
CAC external	[174.69, 2e4, 2e4]	[608.52, 5e4, 5e4]
CAC internal	[1e4, 93.83, 1e4]	[1e4, 310.98, 1e4]
Condenser	[200.59, 2e4, 2e4]	[923.59, 1e5, 1e5]

3.5.4 Physics

Heat exchanger modeling

The porous coefficients in (2.32) are determined using experimental results. Explicitly neglecting convection and viscous fluxes reduces the momentum equation to

$$\frac{\partial p}{\partial x} = -(P^I u^2 + P^V u). \quad (3.1)$$

Applying the ideal gas law to take the changes in the mass flow due to pressure and temperature variations into account and approximating a linear variation of the quantities across the heat exchanger core gives an equation that can be used to obtain the porous coefficients. Given experimental results of pressure drop and temperature variation at different mass flows, the coefficients can be determined by using a least squares fitting approach on

$$\frac{\Delta p}{\Delta x} = - \left(P^I \left(\frac{\dot{m}}{A\bar{\rho}} \right)^2 + P^V \left(\frac{\dot{m}}{A\bar{\rho}} \right) \right), \quad (3.2)$$

where A is the cross-sectional area of the heat exchanger and $\bar{\rho}$ is the mean of the density, calculated using the ideal gas law and linear approximation of the temperature and pressure

$$\bar{\rho} = \frac{p_{mean} M}{RT_{mean}}, \quad (3.3)$$

where M is the molar mass of the gas.

The Matlab file used to determine P^I and P^V can be found in appendix B.1. Coefficients can also be found for the internal regions, using a constant density approach for the coolant in the radiator and approximating the gas in the CAC region as an ideal gas, despite high pressures. The values used in the simulations can be found in table 3.7.

When the porous coefficients were set, the dual stream heat exchanger model was prescribed to the CAC and radiator region. Since the physics inside the condenser is very complex with phase changes, and thereby difficult to model, the single stream approach was used to take the energy transfer from the condenser into account.

The dual stream model requires the user to supply data needed to calculate the local UA_l number, as described in section 2.6.3. In this thesis work the “Actual flow, Q map” option was chosen, since it required quantities supplied by experimental results without any transformation. For more information about the model and which quantities need to be supplied the reader is referred to the aforementioned section.

The single stream model, described in section 2.6.2, needs a prescribed power output and a few other parameters, summarized in table 3.8. For more information about the values the reader is referred to the theory chapter.

Table 3.8: User specified parameters in the single stream heat exchanger model.

Q [W]	ΔT_{min} [°C]	T_{hex} [°C]
7500	5	135

Table 3.9: Specified parameters for the MRF zones.

MRF zone	Origin [mm]	Rotational vector	rpm
Small fan	[1094.4 197.14 572.94]	[1 0 0]	3386.822
Large fan	[1097.75 -138.13 674.42]	[1 0 0]	2761.312

Modeling of the cooling fans

The MRF approach, introduced in section 2.5.1, was used to model the cooling fans. As described, it offers a good trade off between computational demand and accuracy, while at the same time not requiring any mapping to experimental results. The parameters needed to define an MRF zone are summarized in table 3.9, with values taken from the validation case.

Material properties

Specific heat and density of the coolant were set to be temperature dependent. A second order polynomial was used to describe the dependence, with coefficients determined by fitting the polynomial to experimental data. The Matlab code for deciding the coefficients for the density can be found in appendix B.1. The coefficients for the heat capacity polynomial were computed in the same manner.

3.5.5 Solver settings

Supplying sane settings to the solver is crucial in order to achieve converged results in a reasonable time. These settings are very case dependent and are therefore determined ad hoc.

For the present thesis, first order UDS was used for the first 1500 iterations before switching to its second order counterpart. However, switching was not possible for method B, probably due to mesh issues, why first order UDS was used throughout that simulation. Furthermore, the heat exchangers were not activated until iteration 500, in order to improve convergence.

To allow the simulation to develop smoothly, the local Courant number was ramped linearly for the first 300 iterations in the coupled approach; the under-relaxation factors in the segregated solver was treated in the same way.

It was also found that the parameters for the multigrid solver had a significant impact on the convergence behavior. The W-cycle approach was used with two pre- and two post-sweeps with Gauss-Seidel as smoothing method. The meaning of these settings will not be discussed here; the interested reader is referred to an introductory book on the subject (e.g. [1]).

4 Results

In this chapter, results from the validation case will be presented. Focus will be on a comparison between the two methods, with emphasis on the components in the cooling system. When numbers are presented, temperature and pressure are surface averaged, unless stated otherwise.

Note that the second order upwind differencing scheme was used for method A, while convergence issues restricted method B to first order upwinding. This fact alone will explain some of the difference seen between the methods.

4.1 Heat exchangers

The results of the heat exchangers are presented in this section, starting with the two components modeled by the dual stream heat exchanger model, the radiator and the CAC, followed by the single stream modeled condenser.

4.1.1 Radiator

Engineering quantities for the ram air flow through the radiator can be seen in table 4.1. It is noted that the mass flow is slightly lower for method B, which together with a higher inlet temperature explains the lower power output, that in turn explains the lower temperature difference. However, these differences are almost negligible, which also holds for the difference in pressure drop.

Table 4.1: Results for the ram air flow through the radiator.

Method	\dot{m} [kg s ⁻¹]	T_{in} [°C]	T_{out} [°C]	ΔT [°C]	Δp [Pa]	Q [kW]
A	0.939	43.65	96.15	52.54	120.6	49.60
B	0.926	43.95	96.45	52.48	119.9	49.10

Table 4.2 contains the corresponding numbers for the coolant flow in the radiator. All numbers are essentially the same between the methods, much due to that both mass flow and temperature are prescribed at the inlet of the radiator. The tiny discrepancy in outlet temperature, and thus temperature difference, is explained by the small gap in transferred heat.

Table 4.2: Results for the coolant flow in the radiator.

Method	\dot{m} [kg s ⁻¹]	T_{in} [°C]	T_{out} [°C]	ΔT [°C]	Δp [Pa]	Q [kW]
A	2.098	105.25	98.75	-6.46	~ 0	49.60
B	2.098	105.25	97.85	-6.40	~ 0	49.10

The temperature profile for the ram air at the outlet interface of the radiator can be seen in figure 4.1 on the facing page, while the corresponding pressure profile is found in figure 4.2. The temperature profile shows the same trend for both methods, with a higher temperature at the lower part caused by the CAC. The higher temperature at the higher left corner is caused by that the fan shroud is blocking the flow in this region, which also causes the corresponding high pressure zone in figure 4.2. Furthermore, the frozen rotor position is visible in the pressure profile, which is a clear indication on that the requirements on the MRF zone, as presented in section 2.5.1, are not fully satisfied.

Turning to figure 4.5 on page 35, it can be seen that the temperature of the air leaving the condenser, and thus entering the radiator, is higher than the inlet coolant temperature in parts the region heated by the charge air cooler. This implies that the coolant will be heated, instead of cooled, in these regions.

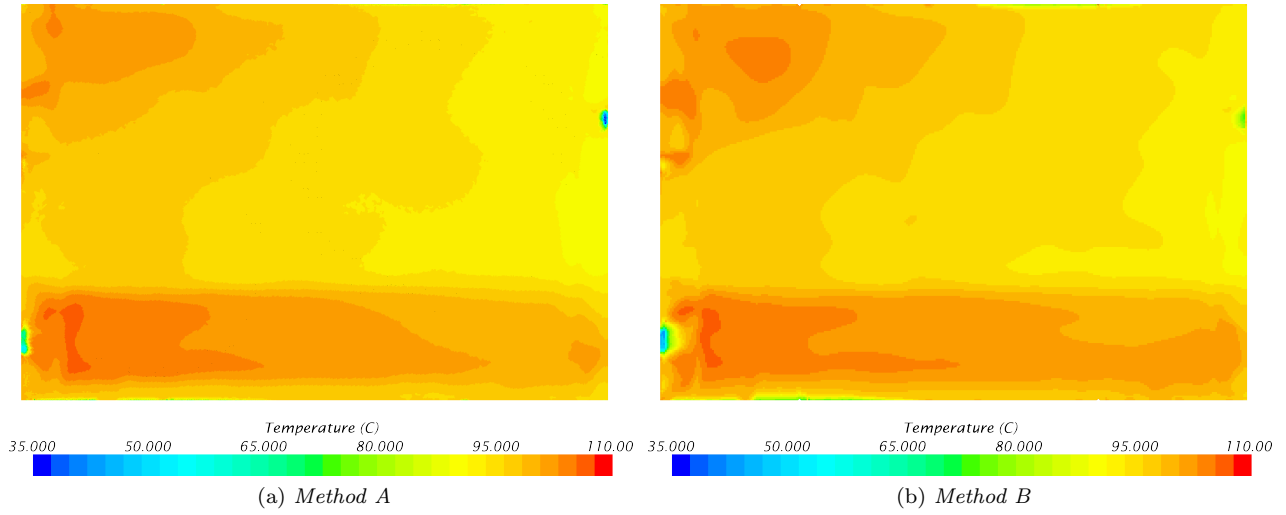


Figure 4.1: *Ram air temperature profiles at the outlet interface of the radiator.*

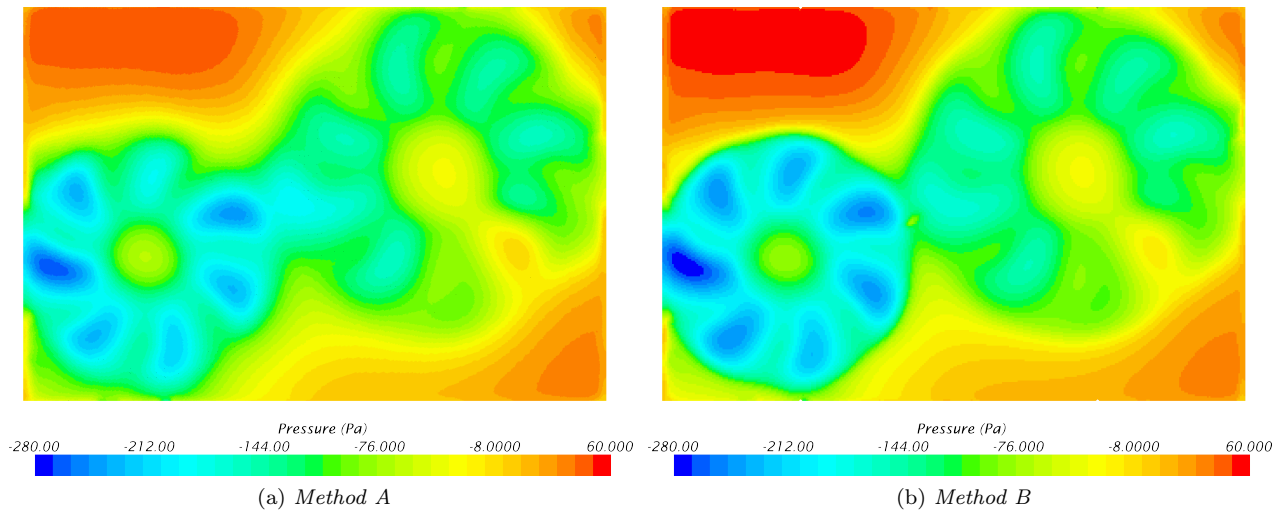


Figure 4.2: *Ram air pressure profile at the outlet interface of the radiator.*

4.1.2 Charge air cooler

Ram air quantities for the charge air cooler is found in table 4.3. It is noted that the inlet temperature is the same between the methods, which is due to the fact that the CAC is mounted in front of the other heat exchangers and thus exposed directly to the non-heated ram air. Since the mass flow is slightly higher for method A, it is expected that the heat output is larger than for method B, which also is the case.

Table 4.3: Results for the ram air flow in the charge air cooler.

Method	\dot{m} [kg s ⁻¹]	T_{in} [°C]	T_{out} [°C]	ΔT [°C]	Δp [Pa]	Q [kW]
A	0.166	27.27	75.53	48.30	115.65	8.102
B	0.160	27.27	75.86	48.60	109.54	7.992

Table 4.4 displays quantities for the charge air flow in the CAC. Just as for the radiator, the mass flow and inlet temperature are essentially the same for the two methods since these quantities are prescribed very close to the heat exchanger core. However, the higher heat transfer rate in method A results in a lower outlet temperature and thus a larger temperature difference, as expected. The small difference in pressure drop is very likely due to a small difference in charge air density caused by the gap in heat output.

Table 4.4: Results for the charge air flow in the charge air cooler.

Method	\dot{m} [kg s ⁻¹]	T_{in} [°C]	T_{out} [°C]	ΔT [°C]	Δp [Pa]	Q [kW]
A	0.099	132.13	50.45	-81.68	1.258e+03	8.102
B	0.099	132.13	51.70	-81.43	1.269e+03	7.992

Ram air pressure and temperature profiles at the outlet interface of the charge air cooler can be found in figures 4.3 and 4.4, respectively. Again, the temperature profiles are similar between the two methods, which also holds for the pressure profiles. However, the low pressure zone is slightly larger for method B, likely caused by the lower pressures seen from the small fan at the outlet interface of the radiator.

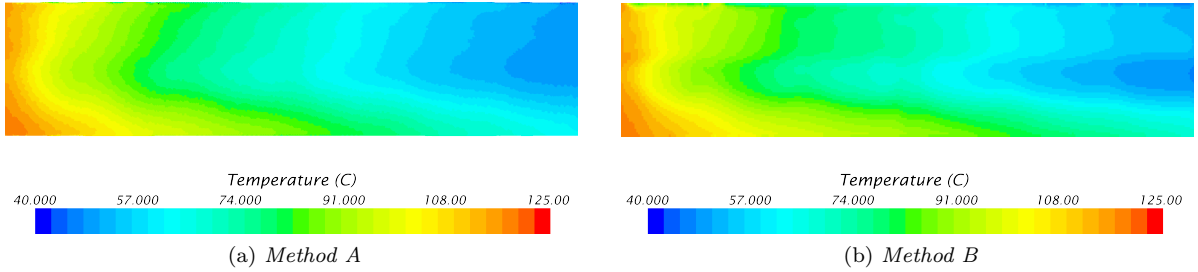


Figure 4.3: Ram air temperature profiles at the outlet interface of the CAC.

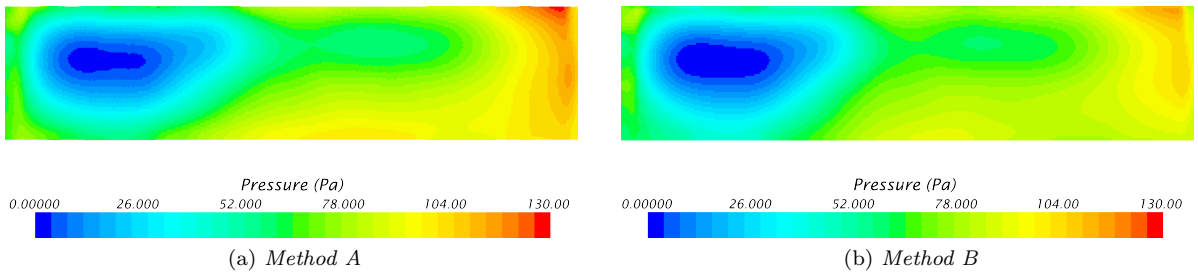


Figure 4.4: Ram air pressure profiles at the outlet interface of the CAC.

4.1.3 Condenser

Since the condenser was modeled with a single stream approach, there are only results for the ram air flow, as presented in table 4.5. The differences between the methods are minor, with a slightly higher inlet temperature for method B caused by its higher outlet temperature from the charge air cooler, which in turn leads to the higher outlet temperature.

Table 4.5: Results for the ram air flow through the condenser.

Method	\dot{m} [kg s ⁻¹]	T_{in} [°C]	T_{out} [°C]	ΔT [°C]	Δp [Pa]
A	0.789	39.65	48.55	8.89	54.14
B	0.787	40.35	49.05	8.75	54.74

Turning to figures 4.5 and 4.6, depicting temperature and pressure profiles at the outlet interface of the condenser, the same trends as for the charge air cooler can be seen. Especially note the aforementioned high temperature zones that give rise to inlet temperatures to the radiator that is higher than the coolant temperature inside it.

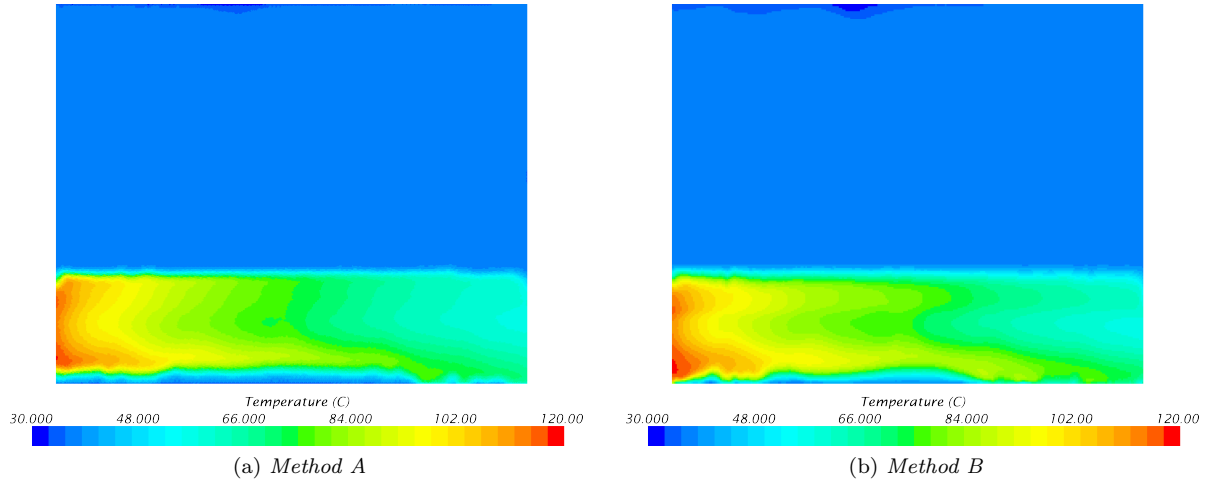


Figure 4.5: Ram air temperature profile at the outlet interface of the condenser.

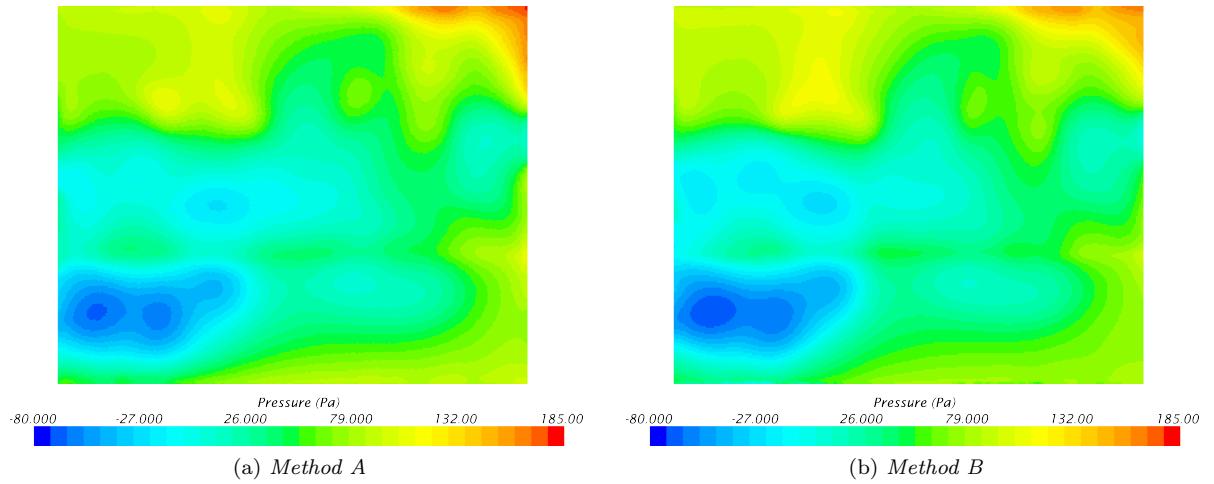


Figure 4.6: Ram air pressure at the outlet interface of the condenser.

4.2 MRF interfaces

Figures 4.7 and 4.8 shows the velocity magnitude at the interfaces of the large and small MRF zones, respectively. Note the white patch in figure 4.7d, which occurs when Harpoon tries to mesh the volume between two close surfaces. If the surfaces are separated by less than two cell sizes, the volume between the surfaces are not meshed. This is not the case when meshing in STAR-CCM+, if the surface wrapper is set to preserve such regions through the contact prevention setting.

Furthermore, the frozen rotor position is clearly visible on the inlet surfaces, indicating that the zone would benefit from being extended here. However, this is not possible due to the surrounding geometry. It can also be seen that the wakes downstream of the blades seems to have mixed properly, which indicates that the MRF zone is sufficiently large here.

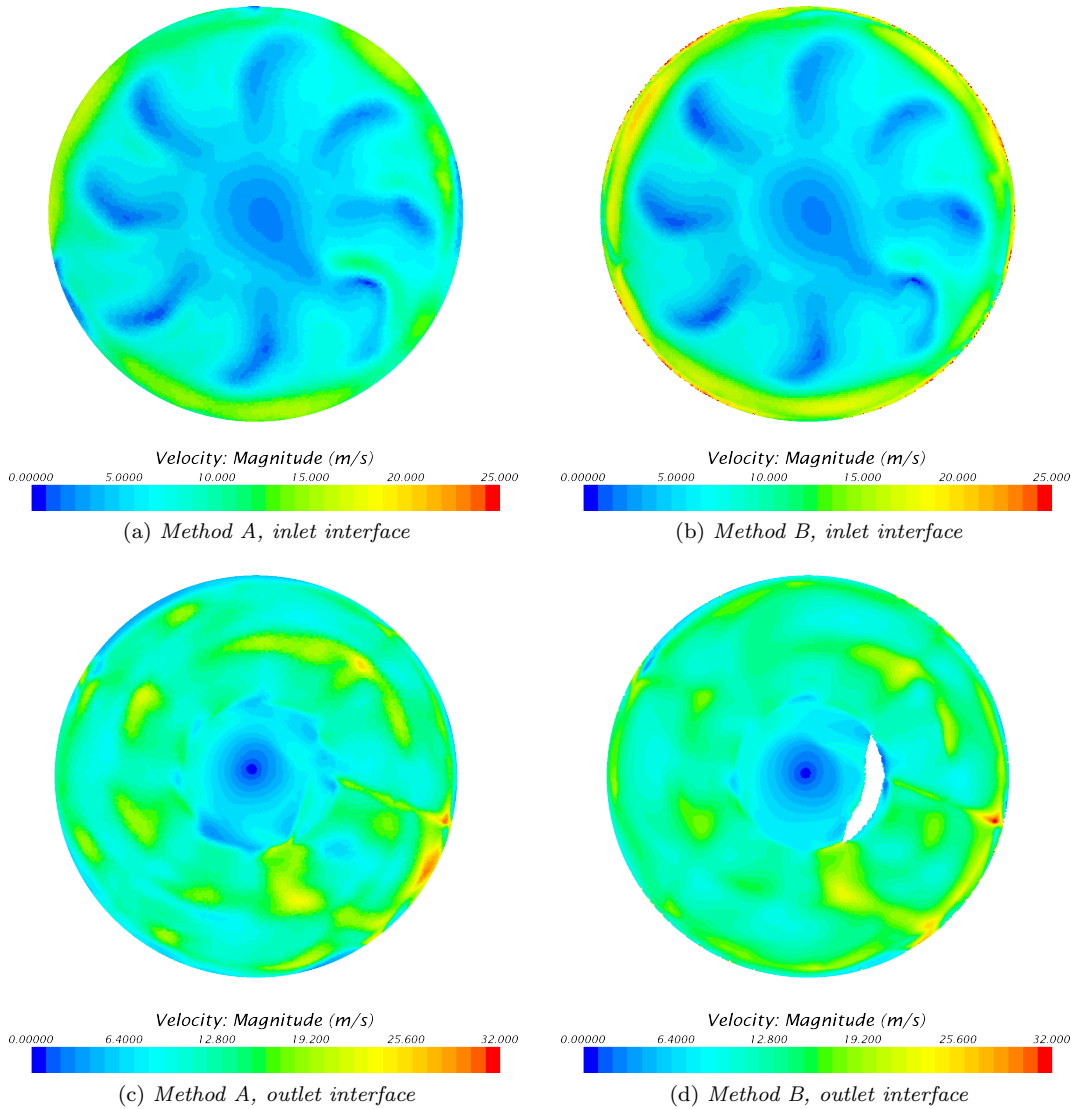


Figure 4.7: Velocity profile at the interfaces of the large MRF zone.

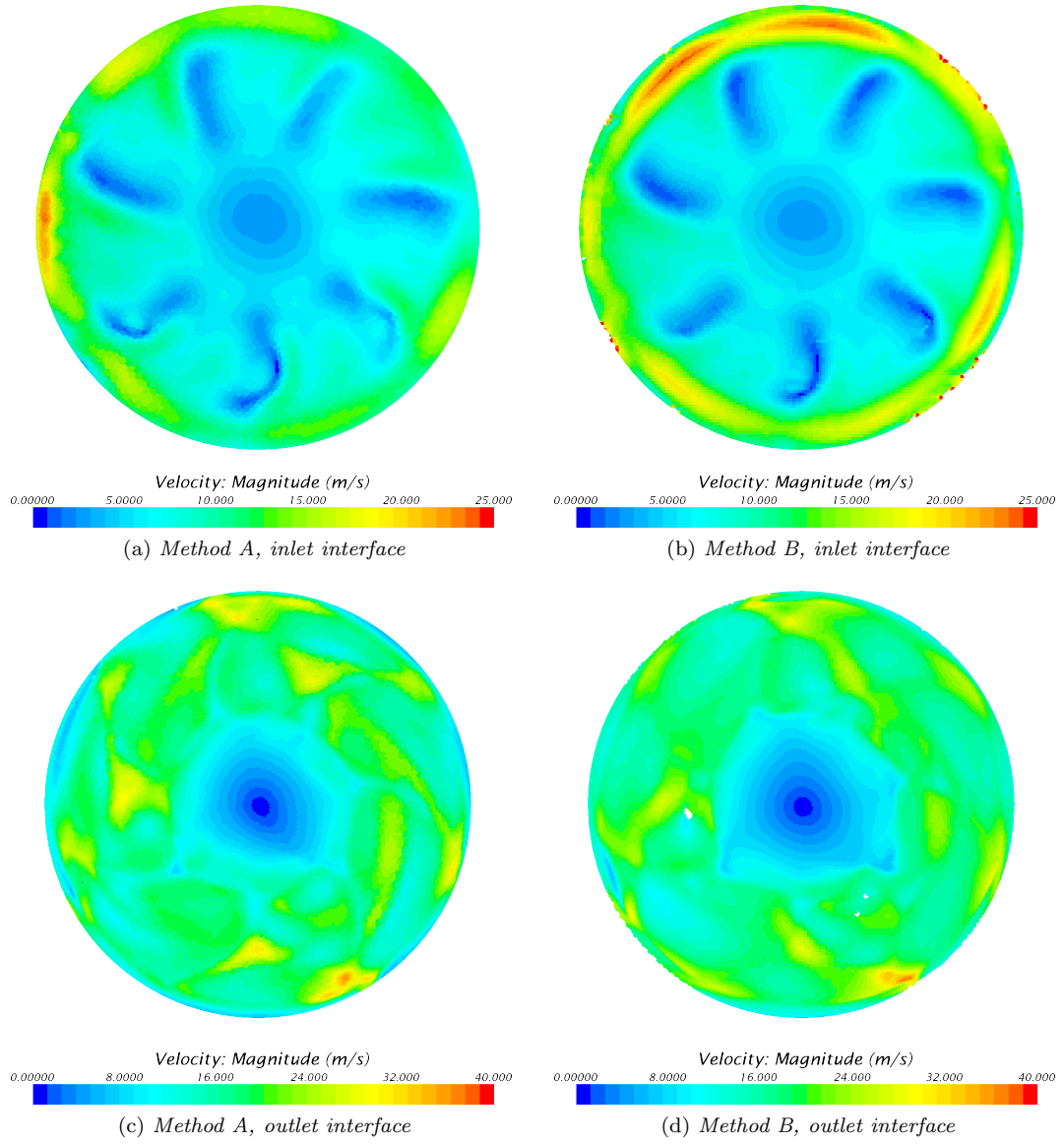


Figure 4.8: *Velocity profile at the interfaces of the small MRF zone.*

4.2.1 Flow behavior

The velocity magnitude on a plane through the car center can be seen in figure 4.9. Comparing the velocity field in the wake of the car in figures 4.9a and 4.9b, a significant difference due to the dissimilar differencing schemes can be seen. Considering the flow in the underhood compartment, the two methods shows the same trends.

Turning to figure 4.9d, it can be seen that the aforementioned regions where the mesh “leaked” has zero velocity. This indicates that they have no substantial contact with the surrounding fluid, which might cause instability or divergence when used with the present uniform initial conditions.

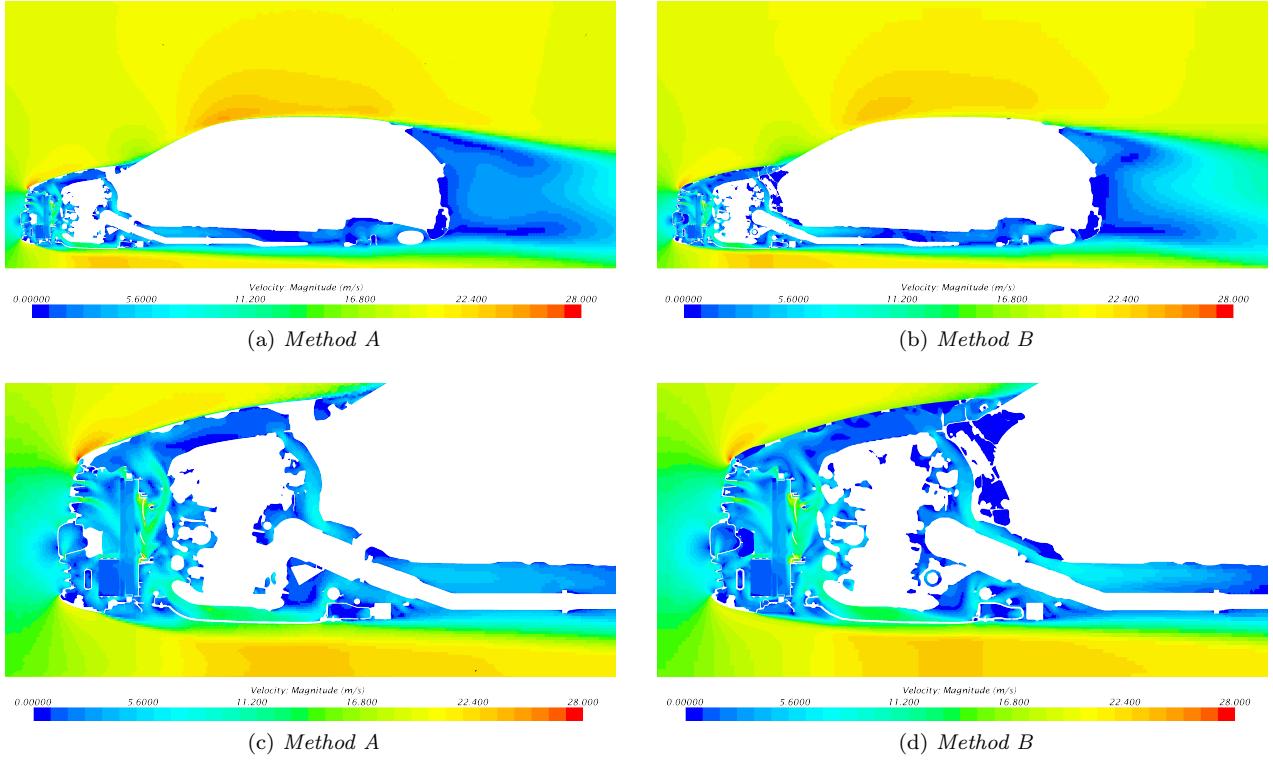


Figure 4.9: Velocity profile at $y = 0$ m. Note that the maximum velocity has been limited for visualization purposes.

5 Discussion

The goal with this chapter is to discuss and compare the two methods and their capabilities and attributes that are important in the current context. Those include:

- Amount of time required to prepare the case.
- Ability to automate the process and manual labor required.
- Validity of the results.
- Numerical stability.
- Computational demand.
- Sensitivity to geometry and parameter changes and ability to handle such in an effortless way.

Since a sensitivity analysis and a comparison between different cases, using the same method, are out of scope of this thesis, the discussion of the last item will be centered around the actions needed to account for changes to an existing case. This is an important feature if the methods are to be used in the concept phase to evaluate different designs of the cooling system and its components.

5.1 Method comparison

Studying table 4.1 to 4.4 on pages 32–34, the heat exchanger results from the two methods can be seen to be similar. The general trend is that method A has a slightly higher ram air flow which will result in a higher heat exchange. This is probably due to a combination of a number of different facts, the most important being that a less accurate differencing scheme was applied and that the constant length surface wrap used to wrap the engine in ANSA gave a slightly larger geometry than the input description. The actual differences in the mass flow and temperature differences only gives rise to about 1.0 % and 1.4 % higher power output for method A of the radiator and CAC, respectively. The difference in temperature and mass flow through the condenser follow the same trend, with 0.2 % and 1.6 % higher mass flow and temperature difference, respectively.

Studying the flow behavior in the underhood region by comparing pressure and temperature profiles on the outlet interfaces of the heat exchangers depicted in figure 4.1 to 4.6 on pages 33–35, the indication that the two methods result in similar results seems to be correct. The profiles have almost identical appearances, with slightly varying gradients in some of the figures, e.g. when comparing the pressure profile in figure 4.4a on page 34 with the corresponding one in figure 4.4b.

As can be seen in figure 3.4b on page 21, the MRF zones are extended above the shroud surface, meaning that the flow should be able to pass through part of the sides of the zones. Applying method A, an interface is readily obtained here by defining the interface between the coincident boundaries in the two regions. For method B on the other hand, the interface creation introduced negative volume cells, preventing interfaces to be created between the regions. This lack of interfaces forces the flow through only the inlet interface in method B, which explains the steep gradients shown close to the edges in figures 4.7a to 4.7b on page 36 and 4.8a to 4.8b on page 37. Some additional effort put into geometry preparation might remedy this problem.

It is hence concluded that based on the results, the two methods show similar capabilities and accuracy. A comparison of the performance of the precursory items will be covered in the remainder this section.

Efficiency

The interesting difference between the two methods in the current context is first and foremost the manual effort needed to prepare the model. Method A has proven to have an advantage because of the fact that all the steps in the preprocessing procedure except for the geometry preparation are carried out in a single software. Even though the surface wrapper in ANSA has great possibilities to customize it to one's own specific needs and reduce the computational time needed to surface wrap the geometry, the time needed to save and communicate the different surface representations between softwares generally increases the time and effort required.

The volume meshing of the two methods both have the advantage of being easily automated. For method B, the automation requires a systematic naming convention and a template batch script that can be modified to the current case, while it for method A is possible both to script the setup in JAVA or use an old simulation

file as a template. Generating the volume mesh (excluding the surface wrap operation) takes approximately 2.5 h (including a surface remesh operation) and 1.5 h for method A and B, respectively. More information about the meshes can be found in section 3.4. For method A, an additional surface remeshing step has to be performed, which takes around 40 minutes.

The computational demand of the two meshes is approximately the same. The polyhedral mesh with prism layers generated in method A consist of fewer cells, but the amount of faces per cell increases the amount of interpolation calculations per cell needed to be carried out. The time required for 3000 iterations using 160 CPUs was approximately 7-8 hours (about 9 seconds per iteration).

Numerical stability

Even though the results are similar between the two simulations, the stability of the solutions are not. Method B requires limiting values of the temperature to avoid unphysical behavior in some regions of the flow. Furthermore, a dissipative first order UDS was applied to increase the numerical stability of the method. Some of the differences in the flow behavior should be caused by the different differencing schemes, as described in section 4.2.1. These stability issues are likely mesh related, since the mesh created in Harpoon had a larger number of low quality cells.

5.2 Validation

Comparing simulation results with wind tunnel data indicates that the power output of the radiator is underestimated by approximately 13 % with the current methods. One possible reason for this poor prediction could be that the temperature of the ram air flow on the inlet interface of the radiator region is higher than the coolant flow in some areas, due to high ram air temperatures out of the CAC. This is seen by studying figures 4.5a to 4.5b on page 35 which depict the outlet temperature profile of the condenser for method A and B, noting that the temperature does not change significantly in the few cells separating the condenser and radiator cores. This fact might cause some of Q_{tot} , calculated from the supplied Q map depending on the mass flow and temperatures of exterior and interior flow through the heat exchanger regions as described in section 2.6.3, to actually be spent to heat the coolant rather than cool it. One way to further investigate this possible cause is to apply the method on a geometry which has the CAC mounted beneath the condenser and radiator rather than in front of them, thus avoiding high air temperatures into the radiator.

Comparing the power dissipation in the CAC, the results seem to show a better prediction when comparing to experimental results. The power dissipation is underestimated by 4.5 % and 6.0 % by method A and B, respectively.

It shall be noted that there is a rather large uncertainty in the temperature measurements in the wind tunnel tests, why the experimental results for the rejected heat can not be assumed to be exact. This means that no firm conclusions can be drawn for the error in the simulations. Furthermore, independent simulations performed at the thermodynamic CFD group at Volvo show similar errors in the heat rejection compared to tests, indicating that there might be issues with the test results for this very case.

5.2.1 Evaluation of the MRF model

The MRF zones are designed according to the theory in 2.5.1. However, despite trying to place the interfaces in regions where the steady state assumption is not violated the velocity profiles can be seen not to be axi-symmetrical. It is hard to improve the design of the zones however, because of the limitations caused by surrounding geometry. The only improvement would be to try to remedy the previously discussed “false baffles” created at the sides of the zones for method B. The errors introduced by the MRF model should be the same between the models and hence not influence the comparison between the models, but rather the comparison between the models and experimental results.

As can be seen in figures 4.7 to 4.8 on pages 36–37, the frozen rotor position is clearly visible in the velocity field on the inlet interface of the MRF zones. This will likely affect the mass flow through the radiator. The larger velocity in some regions will increase the porous resistance more than it will be decreased in the low velocity regions, due to the nonlinearity of the source term in (2.32), which means that the porous resistance will be erroneous, even if the velocity differences cancels.

6 Conclusions

The main conclusions from this thesis can be summarized as:

- CAD cleanup is still time consuming, despite heavy use of surface wrapping.
- The polyhedral mesh created in STAR-CCM+ showed better convergence behavior than its hexahedral counterpart from Harpoon.
- Method A shows significant time savings in terms of manual work compared to method B, due to integration of the meshing into the CFD solver application.
- Harpoon is very insensitive to surface mesh quality. However, the algorithm used has severe problems creating high quality prism layers on complex geometries.
- Heat rejection seems to be underestimated compared to wind tunnel tests. More investigation is needed to determine if the errors lie in the simulation or in the measurements.

Finally, method A was deemed to show the most potential to be incorporated into the workflow at Volvo. The reason is not only better convergence, but also reduced manual time needed to setup an underhood CFD case, compared to method B. The total time to perform a full setup and analysis using method A was estimated to be roughly 30 % shorter than current methods. However, more work is needed before it is ready for production use; for example, conformal interfaces between all regions would likely increase the convergence rate even further.

References

- [1] H. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited, 2007.
- [2] J. Reddy. *An Introduction to Continuum Mechanics*. Cambridge University Press, 2008.
- [3] T. A. Oliver. *Favre-averaged Navier-Stokes and turbulence model equation documentation*. Tech. rep. Predictive Engineering and Computational Sciences, University of Texas at Austin, 2011.
- [4] L. Davidson. *Fluid mechanics, turbulent flow and turbulence modeling*. 2014. URL: http://www.tfd.chalmers.se/~lada/postscript_files/solids-and-fluids_turbulent-flow_turbulence-modelling.pdf.
- [5] T.-H. Shih et al. A new $k-\varepsilon$ eddy viscosity model for high reynolds number turbulent flows. *Computers & Fluids* **24.3** (1995), 227–238.
- [6] L. Davidson. *Numerical Methods for Turbulent Flow. Chapter 5*. URL: http://www.tfd.chalmers.se/~lada/comp_fluid_dynamics/postscript_files/chapter_5.pdf.
- [7] A. Wang, Z. Xiao, and H. Ghazialam. *Evaluation of the Multiple Reference Frame (MRF) Model in a Truck Fan Simulation*. Tech. rep. SAE Technical Paper, 2005.
- [8] P. Gullberg. “Optimisation of the Flow Process in Engine Bays: 3D Modelling of Cooling Airflow”. PhD thesis. Chalmers Tekniska Högskola, 2011.
- [9] CD-adapco. *User guide STAR-CCM+. Version 9.02*. 2014.
- [10] L. A. Piegl and W. Tiller. *The NURBS Book (Monographs in Visual Communication)*. Springer, 1996.
- [11] P. Frey and P.-L. George. *Mesh generation*. John Wiley & Sons, 2010.

A Harpoon batch script

```
1 import tgrid 313B.VED4.msh
2 **VERSION v5.2a7**
3 **MACHINE AMD64_linux_2.6_64**
4 **PREFERENCES USED**
5 **Max Skew 0.999500**
6 **Target Skew 0.980000**
7 **Max Face Warpage 40.000000**
8 **noreset**
9 **intersect**
10 **Separation Angle 40.0**
11 **Setting No. of Cells Between walls to 3**
12 **Setting BDF Exports to Short Format**
13 **Setting BDF Pyramid Treatment to use degenerate PENTA elements**
14 **Setting Max No. Separate Volumes to 10**
15 **Setting No. Cells for Auto Volume Delete to 5**
16 **Setting Part Description to use STL name**
17 **Setting Fluent Thin Wall Treatment to Single Sided**
18 baselev 64.000000
19 farfield global
20 farfield xmin -5000
21 farfield ymin -5000
22 farfield zmin 80
23 farfield xmax 18000
24 farfield ymax 5000
25 farfield zmax 5500
26 wlevel zmin 1
27 **REFINEMENT**
28 ** Refine around the car **
29 refine
30 0 1
31 0 -1500 0
32 8000 1500 2250
33 refexpand 3
34 ** Refine under the car **
35 refine
36 0 2
37 700 -1000 0
38 5500 1000 600
39 refexpand 4
40 ** Refine in the engine bay **
41 refine
42 0 4
43 750 -950 260
44 2250 950 1150
45 refexpand 4
46 **MESH METHODS**
47 type hex
48 expand slow
49 mesh both
50 remove
51 volume -4
52 **SINGLE LEVEL
53 level 2
54 gminlev 1
55 gmaxlev 5
56 plevel *-4 4 4 0
57 plevel *-5 5 5 0
58 plevel *-6 6 6 0
59 plevel *-7 7 7 0
60 pexp *blades* 4
61 pexp *wheel* 4
62 pexp *coolpack* 4
63 **ISOLATE VOLUMES**
64 vfind start
65 wall-*
66 vfind end
67 ** PLUG HOLES **
68 vptplug 3320 0 1050
69 vptplug 3320 0 1900
70 **SORT OUT VOLUMES TO KEEP**
71 **BIG FAN**
72 vnamekeep begin big-fan
73 inter-mrf-large-in-6
74 inter-mrf-large-out-7
75 wall-mrf-large-sides-6
76 blades-large-7
77 vnamekeep end
78 **SMALL FAN**
79 vnamekeep begin small-fan
80 inter-mrf-small-in-6
```

```

81 | inter-mrf-small-out-6
82 | wall-mrf-small-sides-6
83 | blades-small-7
84 | vnamekeep end
85 | **CAC**
86 | vnamekeep begin cac
87 | inter-coolpack-cac-in-6
88 | inter-coolpack-cac-out-6
89 | wall-coolpack-cac-sides-6
90 | intercoolant-coolpack-cactank-in-6
91 | intercoolant-coolpack-cactank-out-6
92 | vnamekeep end
93 | **CAC-IN**
94 | vnamekeep begin cac-in
95 | wall-coolpack-cactankin-sides-6
96 | wall-coolpack-cactankin-in-6
97 | intercoolant-coolpack-cactank-in-6
98 | vnamekeep end
99 | **CAC-OUT**
100 | vnamekeep begin cac-out
101 | wall-coolpack-cactankout-sides-6
102 | wall-coolpack-cactankout-out-6
103 | intercoolant-coolpack-cactank-out-6
104 | vnamekeep end
105 | **COND**
106 | vnamekeep begin cond
107 | inter-coolpack-cond-in-6
108 | inter-coolpack-cond-out-6
109 | wall-coolpack-cond-sides-6
110 | wall-coolpack-condtank-in-6
111 | wall-coolpack-condtank-out-6
112 | vnamekeep end
113 | **RAD**
114 | vnamekeep begin rad
115 | inter-coolpack-rad-in-6
116 | inter-coolpack-rad-out-6
117 | wall-coolpack-rad-sides-6
118 | intercoolant-coolpack-radtank-in-6
119 | intercoolant-coolpack-radtank-out-6
120 | vnamekeep end
121 | **RAD-IN**
122 | vnamekeep begin rad-in
123 | wall-coolpack-radtankin-sides-6
124 | wall-coolpack-radtankin-in-6
125 | intercoolant-coolpack-radtank-in-6
126 | vnamekeep end
127 | **RAD-OUT**
128 | vnamekeep begin rad-out
129 | wall-coolpack-radtankout-sides-6
130 | wall-coolpack-radtankout-out-6
131 | intercoolant-coolpack-radtank-out-6
132 | vnamekeep end
133 | **SMOOTHING MESH**
134 | smooth 2 0.98
135 | smooth 2 all
136 | smooth 2 0.98
137 | vischeck
138 | **save harpoon 313B.VED4.hrp
139 | export ccm+ vol 313B.VED4.ccm

```

B Matlab scripts

B.1 Porous media coefficients

```
1 % Script for fitting of porous resistance parameters for radiators and
2 % charge air coolers.
3 %
4 % According to the STAR-CCM+ User Guide, the pressure drop is given by
5 %
6 %  $dP/dx = \alpha v_n^2 + \beta v_n$ 
7 %
8 % This script fits  $\alpha$  and  $\beta$  to user supplied measurement data from
9 % data sheet. The fitting is done in a least square sense.
10 %
11 % Input:
12 %      deltaP      Vector of pressure losses [Pa]
13 %      deltaT      Vector of temperature differences [K]
14 %      m           Vector of mass flow rates [kg/s]
15 %      P_in        Test condition pressure [Pa]
16 %      T_in        Test condition temperature [K]
17 %      M           Molar mass of the gas [kg/mol]
18 %      width       Radiator core width [mm]
19 %      height      Radiator core height [mm]
20 %      depth       Radiator core depth [mm]
21 %      internal    Boolean. true if internal media, false if
22 %                  external.
23 %
24 % Output:
25 %      Intertial resistance
26 %      Viscous resistance
27
28 clear all
29 close all
30
31 %-----
32 %
33 %                      User defined data
34 %
35 %-----
36
37 % Radiator @ 1kg/s internal mass flow
38 deltaP = [];
39 deltaT = [];
40
41 m = [];
42 P_in = ;
43 T_in = ;
44 M = ;
45 internal = false;
46
47 width = ;
48 height = ;
49 depth = ;
50
51 % CAC @ 0.22 kg/s external. Internal!
52 deltaP = []*1e3;
53 deltaT = [];
54 %
55 m = [];
56 P_in = ;
57 T_in = ;
58 M = ;
59 internal = true;
60 %
61 width = ;
62 height = ;
63 depth = ;
64
65 % CAC @ 0.2 kg/s internal. External!
66 deltaP = [];
67 deltaT = [];
68 %
69 m = [];
70 P_in = ;
71 T_in = ;
72 M = ;
73 internal = false;
74 %
75 width = ;
76 height = ;
```

```

77 % depth = ;
78
79 %-----
80 %
81 %                               Computations
82 %
83 %-----
84
85 % Define the universal gas constant [J/(K*mol)]
86 R = 8.314;
87
88 % Compute upstream density
89 rho_in = P_in*M/(R*T_in);
90
91 % Compute surface area and downstream temperatures and set thickness
92 % according to if the media is internal or external. Convert lengths to [m]
93 if internal
94     A = height*depth*1e-6;
95     thickness = width*1e-3;
96     T_out = T_in - deltaT;
97 else
98     A = height*width*1e-6;
99     thickness = depth*1e-3;
100    T_out = T_in + deltaT;
101 end
102
103 % Compute downstream densities (vector, since it depends on pressure and
104 % temperature)
105 P_out = P_in - deltaP;
106 rho_out = (M/R)*P_out./T_out;
107
108 % Compute the mean temperature and pressure
109 T_mean = 0.5*(T_in + T_out);
110 P_mean = 0.5*(P_in + P_out);
111
112 % Compute density at mean conditions
113 rho = P_mean*M./(R*T_mean);
114
115 % Compute velocity. Convert measurements to [m].
116 v = 1/A*m'./rho';
117
118 % Pack the coefficient matrix
119 coeffs = [v.^2, v];
120
121 % Solve the least squares problem
122 sol = coeffs\'(deltaP/thickness)';
123
124
125 % Print results
126 fprintf('Inertial porous resistance:\t\t%.2f\n', sol(1));
127 fprintf('Viscous porous resistance:\t\t%.2f\n', sol(2));
128
129
130 %-----
131 %
132 %                               Plot
133 %
134 %-----
135
136 % Mass flux for plot
137 m_plot = linspace(min(m), max(m));
138
139 % Convert mass fluxes to velocities
140 v_plot = 1/A.*m./rho;
141
142 % Compute pressure loss
143 dP = (sol(1)*v_plot.^2 + sol(2)*v_plot)*thickness;
144
145 % Plot
146 figure()
147 plot(m, dP)
148 hold on
149 plot(m, deltaP, 'r*')
150 grid on
151 legend('Fitted curve', 'Measurement data', 'Location', 'NorthWest')
152 title('Pressure loss')
153 xlabel('$\dot{m}$ [kg/s]', 'Interpreter', 'LaTeX')
154 ylabel('$\Delta P$ [Pa]', 'Interpreter', 'LaTeX')
155 axis([0 1.1*max(m) 0 1.1*max(dP)])

```

B.2 Coolant density

```
1 % Script for fitting density data to a polynomial of the temperature
2 %
3 % Input:
4 %     T:          Vector of temperatures.
5 %     rho:        Vector of densities of the mixture at given
6 %                temperatures.
7 % Output:
8 %
9 %     Coefficients for 2nd degree polynomial describing the density
10 %    of 50% H2O 50% glycol mixture as a function of the temperature.
11
12 clear all
13 close all
14
15
16 %-----
17 %
18 %                User defined data
19 %
20 %-----
21
22 % 50% volume mixture
23 T = [];
24 rho = []*1000;
25
26
27 %-----
28 %
29 %                Fit data
30 %
31 %-----
32 format shortE
33
34 p = polyfit(T, rho, 2);
35
36 % Print the output
37 fprintf('x^2-coefficient: \t%e\n', p(1));
38 fprintf('x^1-coefficient: \t%e\n', p(2));
39 fprintf('x^0-coefficient: \t%e\n', p(3));
40
41 % Plot
42 Tplot = linspace(min(T), max(T));
43 Cplot = polyval(p, Tplot);
44
45 figure()
46 plot(Tplot, Cplot)
47 hold on
48 plot(T, rho, 'r*')
49 grid on
50 legend('Fitted curve', 'Measurement data', 'Location', 'SouthEast')
51 title('Density')
52 xlabel('T [^\circ C$]', 'Interpreter', 'LaTeX')
53 ylabel('$\rho$ [\frac{J}{kg K}$]', 'Interpreter', 'LaTeX')
54 axis([-40 120 1000 1120])
```

C STAR-CCM+ script for interface creation

```
1  //-----
2  //
3  //      Interface creation macro
4  //      Starccm+ Ver 8.06.005 or later
5  //
6  //      Boundaries intended for interface creation
7  //      should be named either "^inter-.*" or
8  //      "^intercoolant-.*" in order for the macro to
9  //      identify the two corresponding interfaces. If
10 //      the boundaries for the interface creation does not
11 //      have the same names in the two separate regions,
12 //      rename them.
13 //
14 //      In order for the heat exchanger interface to be
15 //      created properly the regions should be named
16 //      ".*-core$" and ".*-core-coolant$".
17 //-----
18 //
19
20 package macro;
21
22 import java.util.*;
23 import star.common.*;
24 import star.meshing.*;
25 import star.base.neo.*;
26 import star.energy.*;
27
28 // Class to store a pair of regions together with a boundary name.
29 // Basically just a very simple container class.
30 class RegionsWithBoundary{
31
32     private String boundaryName;
33     private Region region1;
34     private Region region2;
35
36     public RegionsWithBoundary(String boundaryName, Region region1, Region region2){
37         this.boundaryName = boundaryName;
38         this.region1 = region1;
39         this.region2 = region2;
40     }
41
42     public String getBoundaryName() {
43         return boundaryName;
44     }
45
46     public Region getR1() {
47         return region1;
48     }
49
50     public Region getR2() {
51         return region2;
52     }
53 }
54
55 public class createInterfaces extends StarMacro {
56
57     public void execute() {
58         execute0();
59     }
60
61     private void execute0() {
62
63         // Get the active simulation
64         Simulation simulation_0 = getActiveSimulation();
65
66         // Create collection of regions.
67         Vector<Region> regionColl =
68             (Vector) simulation_0.getRegionManager().getRegions();
69
70         // Initialize a list of lists to store all boundary names for each
71         // region.
72         ArrayList<ArrayList<String>> interBoundNamesArray =
73             new ArrayList<ArrayList<String>>();
74
75         // Loop through the regions to find boundaries with specific names.
76         for (Region region : regionColl) {
77
78             // Create a list to store the the boundary names for the
79             // current region.
80             ArrayList<String> boundNamesInterList =
```



```

81         new ArrayList<String>();
82
83         // Create collection of boundaries for the current region.
84         Collection<Boundary> boundColl =
85             region.getBoundaryManager().getBoundaries();
86
87         // Loop over the boundaries and store the names of the ones
88         // that matches our criteria.
89         for (Boundary boundary : boundColl) {
90
91             // Get the boundary name.
92             String boundName = boundary.getPresentationName();
93
94             // Rename the boundary.
95             boundName = boundName.replaceAll("^.*\\.\\.", "");
96             boundary.setPresentationName(boundName);
97
98             // Check if the boundary name begins with "inter" using
99             // regular expressions.
100             boolean nameCheckInter = boundName.matches("^inter.*");
101
102             // If so, store the name.
103             if (nameCheckInter) {
104                 boundNamesInterList.add(boundName);
105             }
106         }
107
108         // Add the list of matching boundary names for this region
109         // to the list of lists containing the matched names
110         // for all regions.
111         interBoundNamesArray.add(boundNamesInterList);
112
113         // Clear the intermediate list to avoid false positives in the
114         // next region.
115         boundColl.clear();
116
117         // Create heat exchanger interfaces.
118         // Get the presentation name of the current region.
119         String regionName = region.getPresentationName();
120
121         if (regionName.matches("^.*core$")) {
122
123             // Get the corresponding coolant region.
124             Region regionCoolant = simulation_0.getRegionManager().
125                 getRegion(regionName + "-coolant");
126
127             // Create and rename the h.ex. interface.
128             DirectRegionInterface hexInterface = simulation_0.getInterfaceManager().
129                 createDirectRegionInterface(region, regionCoolant, "Heat-Exchanger");
130             hexInterface.setPresentationName(regionName + "-regionInterface");
131
132             // Set the dual stream model to be "Actual flow
133             // dual stream" and use the "q map" method.
134             hexInterface.getConditions().get(HeatExchangerMethod.class).
135                 setSelected(HeatExchangerMethod.ACTUAL_FLOW_DUAL_STREAM);
136             hexInterface.getConditions().get(HxActualFlowDataOption.class).
137                 setSelected(HxActualFlowDataOption.ACTUAL_FLOW_Q_MAP);
138
139         }
140     }
141 }
142
143
144 // Create a list of RegionsWithBoundary objects to store the interface
145 // candidates in a single container.
146 ArrayList<RegionsWithBoundary> boundaryMap =
147     new ArrayList<RegionsWithBoundary>();
148
149 // Loop over all regions, except the last.
150 for (int i = 0 ; i < (regionColl.size()-1) ; i++) {
151
152     // Loop over all regions "in front of" the current (base)
153     // region in the list.
154     for (int j = i+1 ; j < regionColl.size() ; j++) {
155
156         // Loop over all boundary names in the base region.
157         for (String intermediate : interBoundNamesArray.get(i)) {
158
159             // Check if the boundary name exists
160             // in another region.
161             if (interBoundNamesArray.get(j).contains(intermediate)) {
162
163                 // Create a RegionsWithboundary to store

```

```

165                                     // the two regions and the boundary name.
166 RegionsWithBoundary rwb = new RegionsWithBoundary(
167     intermediate, regionColl.get(i), regionColl.get(j));
168
169                                     // Insert the RegionsWithBoundary into
170                                     // the boundaryMap
171 boundaryMap.add(rwb);
172
173                                     }
174
175     }
176
177 }
178
179 }
180
181 // Loop over all interface candidates in order to create interfaces.
182 for (RegionsWithBoundary rwb : boundaryMap) {
183
184     // Find the boundaries needed to create the interfaces.
185     // Use the get methods in RegionsWithBoundary.
186     Boundary boundary_0 = rwb.getR1().getBoundaryManager().
187         getBoundary(rwb.getBoundaryName());
188     Boundary boundary_1 = rwb.getR2().getBoundaryManager().
189         getBoundary(rwb.getBoundaryName());
190
191     // Create the interface. Give it the same name as the boundary.
192     DirectBoundaryInterface directBoundaryInterface_0 = simulation_0.
193         getInterfaceManager().createDirectInterface(boundary_0, boundary_1,
194             rwb.getBoundaryName());
195
196     // Set the interface type.
197     directBoundaryInterface_0.getTopology().
198         setSelected(InterfaceConfigurationOption.IN_PLACE);
199
200     // Set presentation name.
201     directBoundaryInterface_0.setPresentationName(rwb.getBoundaryName() + "-interface");
202
203 }
204
205 }
206

```