2015-11-01

Aerodynamic Improvement of the BYU Supermileage Vehicle

Sayan Dobronsky
Brigham Young University - Provo

Aerodynamic Improvement of the BYU Supermileage Vehicle

Sayan Dobronsky

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

November 2015

ABSTRACT

Aerodynamic Improvement of the BYU Supermileage Vehicle

Sayan Dobronsky
Department of Mechanical Engineering, BYU
Master of Science

Engineers have been working to improve road vehicle fuel efficiency through aerodynamics for decades. Competitions such as the Society of Automotive Engineers (SAE) Supermileage series and the Shell Eco-Marathon, in which BYU participates, encourages students to build and drive a supermileage vehicle with the main goal of achieving the highest possible fuel efficiency. Overcoming aerodynamic drag is one of the challenges to improving the vehicle's fuel efficiency.

The purpose of this thesis work was to design a new shape for the BYU Supermileage vehicle in order to improve its fuel efficiency. Computational Fluid Dynamics (CFD) was used to obtain the coefficient of drag (C_D) and drag area of the current baseline vehicle at a Reynolds number of 1.6×10^6 and 8.7×10^5. Then a new shape was developed using mesh morphing software. The new shape was imported into the CFD program and the drag figures and airflow plots from the modified design were compared with the baseline vehicle. Scale models of the vehicles were also printed using a 3D printer in order to perform wind tunnel testing. The models were installed in the wind tunnel and the coefficient of drag and drag area were compared at a Reynolds number around 8.7×10^5.

It was found from the CFD results that the new vehicle shape (labelled Model C) caused a 10.8% reduction in C_D and a 17.4% reduction in drag area under fully laminar flow. Smaller drag reductions were observed when the flow was fully turbulent. From the wind tunnel comparisons, it was found that Model C reduced C_D by 5.3% and drag area by 11.4%, while the fully laminar CFD results at Re = 8.7×10^5 showed that Model C reduced C_D by 9.8% and drag area by 15.9%. Smaller drag reductions were again observed for fully turbulent flow. Thus in order to improve the aerodynamic performance, the current vehicle shape should be changed to match that of Model C, and laminar flow should be encouraged over as much of the wetted area as possible.

Keywords: supermileage vehicle, CFD simulations, aerodynamics, drag reduction, wind tunnel

ACKNOWLEDGMENTS

I am grateful to the mechanical engineering department here at BYU for generously providing funding for my education. I am especially grateful for my committee chair, Dr. Bowman, for bringing to my attention the exciting idea of working on the aerodynamics of the Supermileage vehicle. His guidance throughout this project has been absolutely critical, and his willingness to consider my ideas and work with me as I left school for multiple internships was invaluable. I am also grateful to my graduate committee, Dr. Gorrell, and and Dr. Ning, for their help and consideration in this project.

I am also grateful to the graduate students in the Turbomachinery lab, the Fluids and Heat Transfer lab, and the students who were in the Splash lab. I would often go to these different labs with many questions, and the students were more than willing to take time to address my questions and concerns and to brainstorm with me.

I would also like to thank my loving parents. There are no words to describe the support that they have continually offered me, and even now from hundreds of miles away, their support means everything.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

C_D	Coefficient of Drag
C_L	Coefficient of Lift
C_P	Pressure Coefficient
D	Aerodynamic Drag
V	Velocity
φ	Blockage Ratio
A	Frontal Area of Vehicle
ΔP	Pressure Difference
ρ	Air Density
Abbreviations	
ABS	Acrylonitrile Butadiene Styrene
ASD	Arbitrary Shape Deformation
CFD	Computational Fluid Dynamics
DOE	Design of Experiments
FSL	Fulton Supercomputing Laboratory
LVDT	Linear Variable Displacement Transducer
MPG	Miles per Gallon
NACA	National Advisory Committee for Aeronautics
PLA	Polyactic Acid
SMV	Supermileage Vehicle
Re	Reynolds Number

CHAPTER 1. INTRODUCTION

Automobile fuel efficiency has been a major area of research for decades. However, in recent years the effort to increase fuel efficiency has been aided by an increased demand by the public for vehicles that are fuel efficient, and by advances in technology that enable cars to travel longer distances on less fuel. Today, almost every major vehicle manufacturer offers some type of hybrid vehicle. The federal government has also taken an interest in increasing fuel efficiency by mandating improvements to the average miles per gallon (MPG) rating of a manufacturer's lineup of cars and trucks (see `www.nhtsa.gov/fuel-economy`). With the increased attention to fuel economy, engineers have stepped up their efforts to leverage new technologies to satisfy the demand for more fuel efficient vehicles. These efforts have led to the development of a vast array of technologies, including everything from plug-in electric vehicles to fuel cells. In 2005, a team of engineers from ETH Zurich developed a supermileage vehicle named PacCar II that achieved 12,666 MPG, a world record for fuel efficiency [1]. A photo of PacCar II is shown in Figure 1.1.

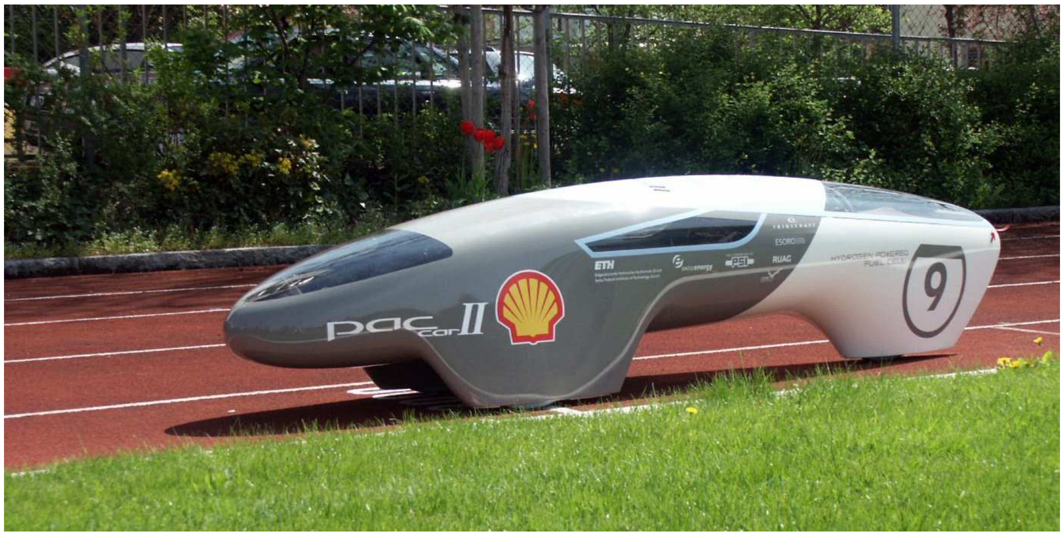

Figure 1.1: Photo of PacCar II

PacCar II was run at the Shell Eco-Marathon, a competition designed to motivate students and engineers to build vehicles to achieve the highest MPG rating. This competition, along with others such as the Society of Automotive Engineers (SAE) Supermileage, (part of the SAE collegiate design series) are part of the increased effort to develop new technologies and innovative methods to create more fuel efficient vehicles. One of the factors that led PacCar II to achieve such a high MPG rating was its aerodynamic performance. The vehicle won the European Car Aerodynamics Research Association (ECARA) award in 2007 for its aerodynamic design (see `http://www.ecara.org/dateien/ECARA_Award_2007_Laudatio.pdf`). Vehicle aerodynamics has been an active area of research for many years and has once again received attention as part of the effort to increase fuel efficiency.

1.1 The BYU Supermileage vehicle

BYU has been competing in both the SAE Supermileage and Shell Eco-marathon competitions with its own car developed by senior capstone teams every year. The design of the BYU Supermileage vehicle is driven by its main objective of achieving the highest possible fuel efficiency. This objective affects everything from shape design to the selection of a driver. The shape of the vehicle is based on the shape of PacCar II in an attempt to keep its aerodynamic drag low. The vehicle has two front wheels that turn, and one rear wheel that is driven by a modified Briggs & Stratton engine. Figure 1.2 shows a photo of the 2015 vehicle.

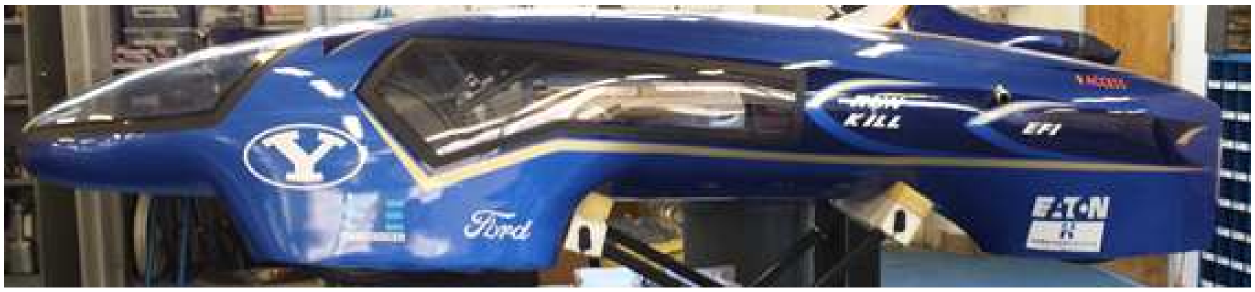

Figure 1.2: Photo of BYU Supermileage Vehicle

1.1.1 Problem Statement

With the objective of the BYU Supermileage vehicle in mind, it is helpful to formulate a problem statement for this thesis in order to guide the work. The question can be asked: How can we improve the fuel efficiency of the Supermileage vehicle through aerodynamics? This problem statement helps to put the focus of the work on aerodynamic improvements, as opposed to other avenues such as improving engine efficiency or weight reduction. Thus the bulk of this thesis work will focus on investigating how the aerodynamic drag of the vehicle can be reduced.

1.2 Literature Review

There exists a vast array of technical and non-technical papers covering the field of road vehicle aerodynamics. However, there are fewer covering the aerodynamic performance of Supermileage vehicles. Hucho *et al.* [2] provides one of the most comprehensive works covering aerodynamics of road vehicles in general. Topics covered include car, truck, and motorcycle aerodynamics, engine cooling, directional stability, wind noise, and high performance vehicle aerodynamics. Of particular interest are the topics of detail and shape optimization, where the basic shell of a vehicle is modified slightly in an effort to lower the drag coefficient. For detail optimization, the basic concept of the vehicle shape cannot be changed by the aerodynamicist. Only small details are allowed to be altered, such as the edges of pillars, tapering, and size of spoilers. This method provided reductions in drag coefficients of cars but soon reached its limits as car stylists anticipated the impact of their designs on aerodynamics. Shape optimization, on the other hand, allows the aerodynamicist to start with a volume called the ”basic body” which has the same critical dimensions as the final car shape. As the car undergoes the development process, the basic body is changed through multiple steps until a ”basic shape” emerges, which serves as the design for the car.

Regarding Supermileage vehicles, Maji *et al.* outlined their development process for designing a new shape for their vehicle [3]. Their team started with 2D airfoil shapes to design a new vehicle shell based off of an inverted tricycle shape, with two wheels at the front and one at the rear, similar to the BYU Supermileage vehicle. Computational Fluid Dynamics (CFD) analysis was done with a volume mesh of 389,575 tetrahedral cells. Wind tunnel tests were also performed with a 1:5 scale model of the vehicle shape. Flow visualization was added with paint on the surface and the dryness pattern was observed as the paint was drying, as shown in Figure 1.3. The team noted that ”considerable improvement” was made with the drag results as compared to the previous year’s vehicle, although the drag numbers for the previous year’s vehicle were not given.

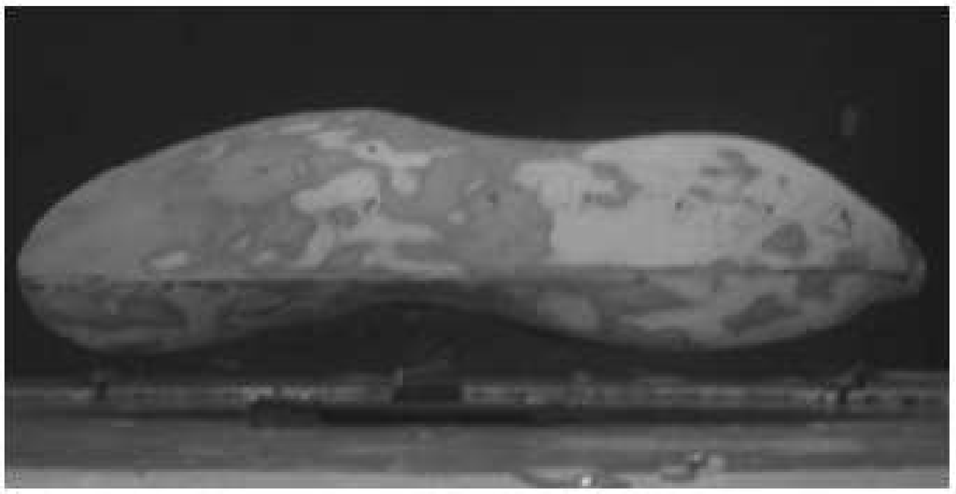

Figure 1.3: Photo of Flow Visualization on 1:5 Scale Model from [3]

Gagnon *et al.*, from the University of Laval, also designed a shape for their team’s Supermileage vehicle [4]. Their team used NACA 3312 airfoil shapes to modify the old vehicle design. Other vehicle considerations, such as reinforcing the wheel fairings, forced the team to increase their frontal area, which would normally increase aerodynamic drag. Despite the frontal area increase, the team used CFD analysis, with an emphasis on studying the airflow around the nose and tail of the vehicle, to decrease the drag coefficient by 20%, although the main metric that was monitored through the design process was energy usage per meter driven. Figure 1.4 shows their front nose drag picture. The top of the original vehicle was an ellipse tilted forward, but the new vehicle top used the NACA 3312 airfoil shape as a guide.

The development process for PacCar II (shown in Figure 1.1) was described by Santin *et al.* [1]. Their process for designing an aerodynamically efficient shape was similar to the shape optimization process described by Hucho. CFD analysis and wind tunnel testing on a 1:2 scale model were done to study the airflow around the vehicle. The effect of surface treatments on drag was also discussed. Their drag numbers are compared in Chapter 2.

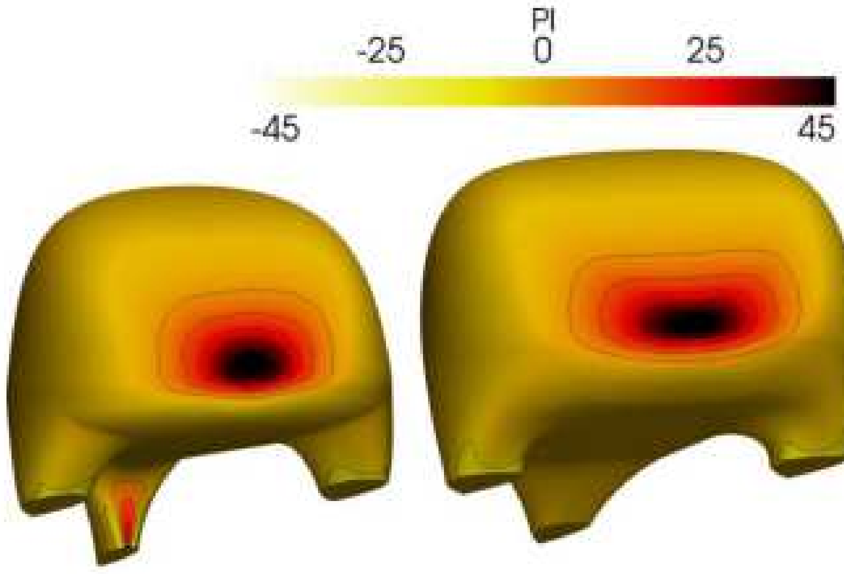

Figure 1.4: Drag Distribution of Old (left) and New (right) Shells of Laval SMV from [4]

1.3 Research Objectives

The objective of this thesis work is to design a new shape for the BYU Supermileage Vehicle (SMV) in order to decrease its aerodynamic drag to further improve its fuel efficiency. During the early stages of research, the decision was made to modify the current baseline design in order to develop a new shape. Because of this, a full optimization study was not undertaken, but some design of experiments (DOE) studies were done to study specific sections of the vehicle. The resulting design will be shown to decrease aerodynamic drag using tools such as CFD simulations and wind tunnel testing. The new shape will be used for the next iteration of the vehicle built by the 2015-16 capstone team.

The aerodynamic drag of a vehicle, D, is described by the following equation:

$$D = \frac{1}{2}\rho V^2 A C_D \tag{1.1}$$

where ρ is the density of the air surrounding the vehicle, A is the vehicle frontal area, V is the vehicle speed, and C_D is the coefficient of drag. Thus the drag is affected by the vehicle shape (reflected by the value of C_D) and its frontal area (reflected by the value of A). Since modifications were made to both the shape and the frontal area in search of a more aerodynamic vehicle, the drag area, defined as C_DA, was chosen as the parameter that would define the aerodynamic performance of the vehicle. Therefore the objective was to reduce the drag area by defining a new vehicle shape.

1.4 Overview of Chapters

1.4.1 Chapter 2

Chapter 2 covers the use of CFD and mesh morphing software to design a new shape for the BYU SMV. The process of modifying the shape through Arbitrary Shape Deformation (ASD) volumes in Sculptor is outlined. The CFD simulation process is covered, including model import, meshing, physics setup, simulating, and post-processing. Solution plots from the CFD simulations are shown and comparison is made between the current baseline vehicle and the new modified shape. Comparison is also made with published data from PacCar II engineers.

1.4.2 Chapter 3

Chapter 3 covers the experimental process of wind tunnel testing. The process of generating CAD models for 3D printing along with the 3D printing process is outlined. The experimental setup is described and results for all models tested are shown.

1.4.3 Chapter 4

Chapter 4 presents a discussion of the results from the CFD simulations and the experimental trials. The results are compared and discrepancies are discussed, along with the impact of the improved aerodynamics on the fuel efficiency of the new vehicle.

1.4.4 Chapter 5

Chapter 5 summarizes the conclusions from chapter 4 and combines them with an overview of the work that encompasses this thesis. Recommendations are then given in order to lower the aerodynamic drag on the vehicle. Possibilities for further research are also discussed.

CHAPTER 2. NUMERICAL SIMULATIONS

2.1 CFD Methods Overview

In order to use CFD to quantify the aerodynamic drag on the baseline and modified Supermileage vehicle, a CAD or mesh model representing the vehicle shape had to be imported into the CFD program. Figure 2.1 shows the work flow for obtaining the drag area of a vehicle from CFD simulations, beginning with a model that represents the shape of the vehicle. The CFD program that was used for all simulations was Star CCM+, version 9.02.007.

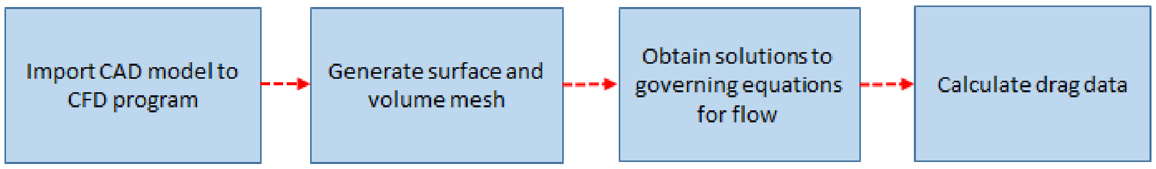
Import CAD model to CFD program
Generate surface and volume mesh
Obtain solutions to governing equations for flow
Calculate drag data

Figure 2.1: Process of Obtaining Drag Area from CFD Simulations

2.2 CFD Settings

The following sections outline the many settings that needed to be specified in order to obtain drag information on the vehicles. These settings include the fluid domain size, the mesh parameters, the physics settings, the script commands for running simulations on the BYU Supercomputer, and post processing settings.

2.2.1 Model Import and Fluid Domain

The import options available on Star CCM+ allowed for the import of both a CAD model and mesh model of the vehicle. For the baseline vehicle, a CAD model generated by a previous capstone team was available to use. For modified vehicle shapes, the surface mesh of the vehicle was used for importing. Once the model was imported, a rectangular geometry had to be created that would serve as the virtual ”wind tunnel” in which the vehicle would be experiencing airflow. The dimensions of the virtual wind tunnel which contained the fluid domain depended on the size of the vehicle; the fluid domain was created large enough so that effects from the walls would be negligible. Recommendations from CD Adapco training materials were used as a guideline to dimension the fluid domain [5]. These recommendations were developed originally from simulations on a generic Formula SAE race vehicle, and were adapted as necessary for the Supermileage vehicle. Figure 2.2 shows the minimum fluid domain dimensions for the Supermileage vehicle. Often bigger dimensions were used to give space for further mesh refinement near the vehicle. As Figure 2.2 shows, the domain inlet was placed five body lengths upstream of the vehicle, the

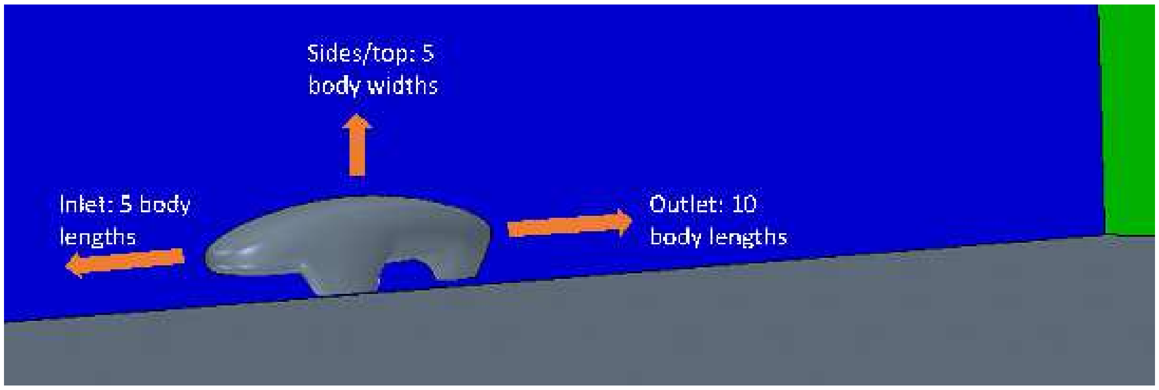

Figure 2.2: Dimensions of the Fluid Domain in Terms of Body Lengths

domain exit was placed ten body lengths downstream, and the domain top and sides were placed five body widths away from the vehicle. The fluid domain was also created so that a symmetry plane was placed at the midpoint of the vehicle, with only half of the vehicle in the fluid domain. Utilizing a symmetry plane saved computing time since a surface and volume mesh covering the entire vehicle was not needed, and the governing flow equations needed to be solved for only half the volume as compared to a fluid domain enclosing the full size vehicle. Once the geometry was set, the various boundaries of the fluid domain were declared to be walls, symmetry planes, velocity inlets, or pressure outlets. For example, the inlet of the domain was specified to be a velocity inlet so that a uniform flow normal to the inlet face could enter the fluid domain.

2.2.2 Mesh Settings and Grid Independence

Training materials from CD Adapco provided a guideline for the mesh settings used for the baseline vehicle [5]. These settings were also adapted as necessary for the size and shape of the baseline vehicle and any modified vehicles. The surface remesher option was used to generate the surface mesh of the imported model. This covered the surface with triangular elements that defined the grid points where the flow solutions would be calculated. The training material recommended either the trimmer or the polyhedra for the volume mesh, and the trimmer mesh was selected to take advantage of the trimmer wake refinement feature. Figure 2.3 shows close up views of the surface and volume meshes on the front of the baseline vehicle.

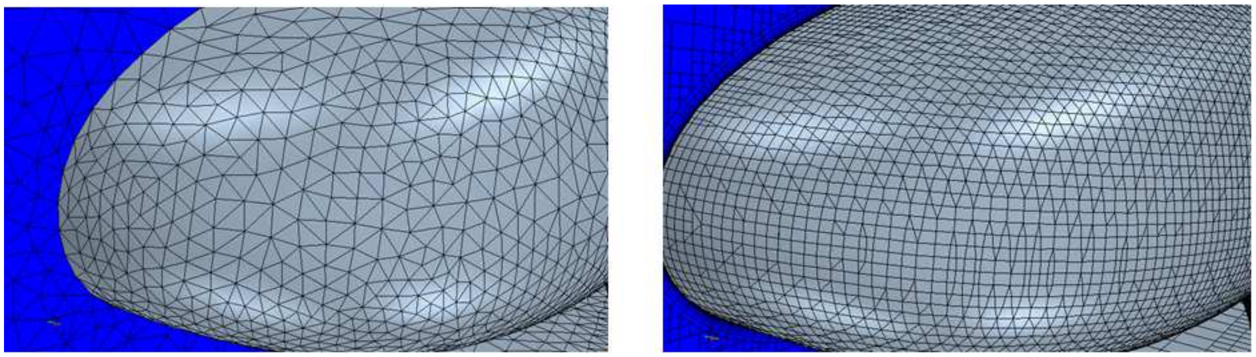

Figure 2.3: Close up View of the Polygonal Surface (left) and Trimmer Volume Mesh (right) of Baseline Vehicle

Most of the surface and volume mesh settings could be connected to the base size, which is the typical size of a cell in the mesh. Almost all other mesh settings, such as the maximum cell size, and the volumetric refinements, could be set as a percentage of the base size. Using a base size made it simple to change the mesh as needed; only the base size needed to be changed in order to increase or decrease the grid spacing. For the baseline vehicle, the base size was set to 14 mm, which was similar to recommendations from CD Adapco. In order to accurately resolve the flow characteristics in the region close to the vehicle, two volumetric controls enclosing the vehicle were specified. The smaller of the two controls was named ”near field” and restricted the maximum cell size to be 500% of the base size. The larger control was named ”mid field” and restricted the cell size to 1000%. In this manner the cell sizes would not grow too large to inaccurately model the flow in the vicinity of the vehicle. A trimmer wake refinement was also specified behind the vehicle.

Similar to the volumtetric controls, the wake refinement allowed for control over the base size in a region extending behind the vehicle. The cell size was restricted to 50% of the base size in the wake region. Figure 2.4 shows the mid field and near field grids, along with the wake refinement. Prism layers were also added to the surfaces of the vehicle in the volume mesh in order to capture

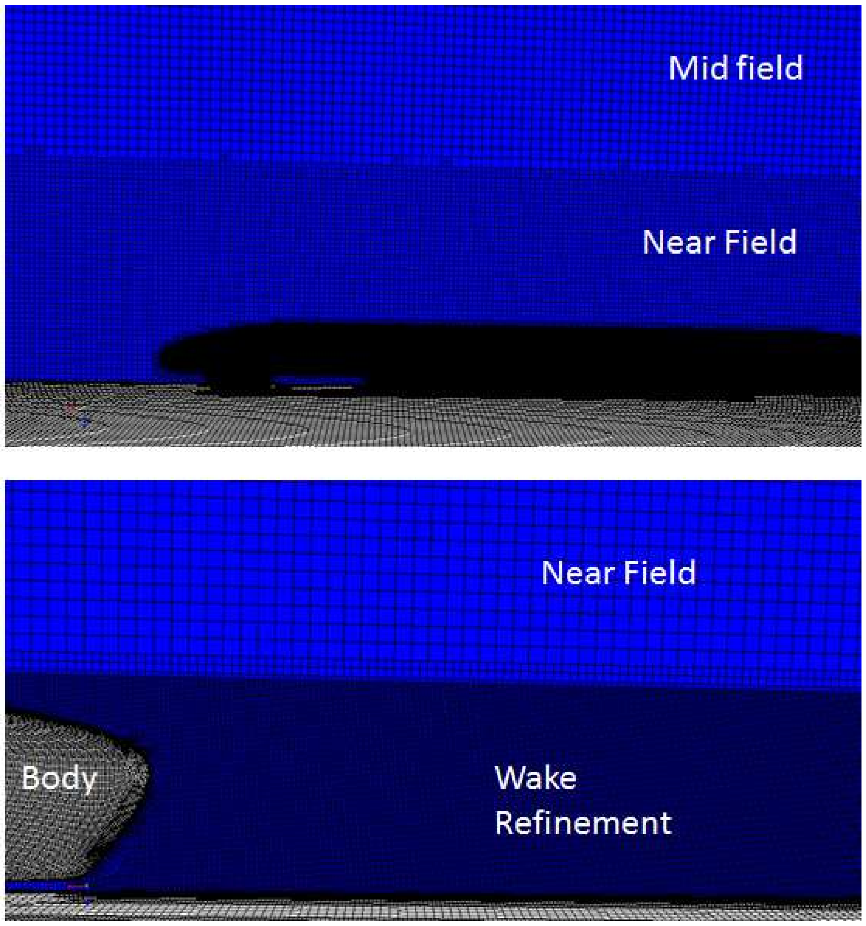

Figure 2.4: Volumetric Refinements near the Body of the Vehicle

the flow in the boundary layer. Prism layers are smaller cells that are added on the surface of a body in order to resolve the boundary layer characteristics. Typically the number of prism layers along with the total prism layer thickness is specified in order to generate the required number of cells near the surface. Due to the curvature of the rear portion of the vehicle, extra prism layer controls needed to be changed in order to force the placement of prism layers at the rear. This included changing the boundary march angle, the concave and convex angle limit, and the near core layer aspect ratio. The resulting volume mesh then contained prism layers over the entire boundary of the body surface. 12 prism layers were used, with a total prism layer thickness of 24 mm. These values were similar to the ones found in the guidelines for vehicle meshing from CD Adapco [5]. Figure 2.5 shows a closeup view of the prism layers.

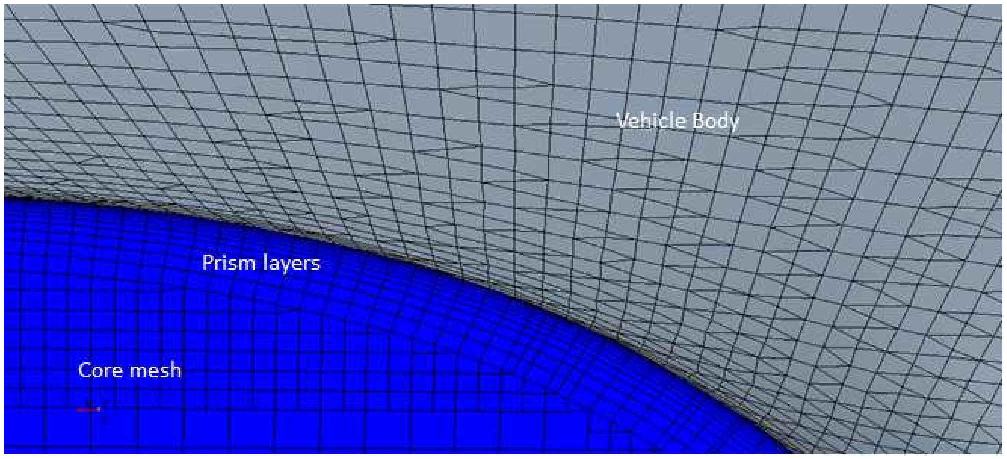

Figure 2.5: Closeup View of Prism Layers Near Underbody of Vehicle

The total number of cells in the entire fluid domain was based on the results of a grid sensitivity analysis. This was done in order to ensure that the value for the coefficient of drag and lift was not dependent on the size of the grid spacing. In other words, the solutions generated from the simulations needed to be grid independent. First an initial grid spacing was chosen by specifying the base size, and the coefficient of drag was recorded after the simulation completed. Then the grid spacing was reduced (usually halved) in order to increase the total number of cells, and the coefficient of drag was recorded again and compared to the last simulation. The solution was considered to be grid independent once the coefficient of drag was no longer changing drastically between simulations. The change in coefficient of drag that was permitted before concluding grid independence was based upon a tradeoff between increased accuracy with a finer mesh, and the resulting increase in meshing and solution time. Figure 2.6 shows the change in the coefficient of drag as a function of grid size for the baseline vehicle under laminar flow. Since the grid sensitivity analysis was done early in the CFD process in order to find an appropriate grid size for simulations, the resulting values for C_D do not reflect the final values used for the CFD results, since some fine tuning of the mesh was done after the grid sensitivity analysis. The graph shows that the change in C_D decreases as the mesh becomes finer, with a leveling off that occurs after 7×10^6 cells. Considering the increased time in generating a solution for grids with more than 7×10^6 cells, it was decided that 7×10^6 cells was enough for grid independence. Thus the baseline simulations used a mesh size of 7×10^6 cells, and most of the modified vehicle simulations used a similar mesh size of 6.7×10^6 cells.

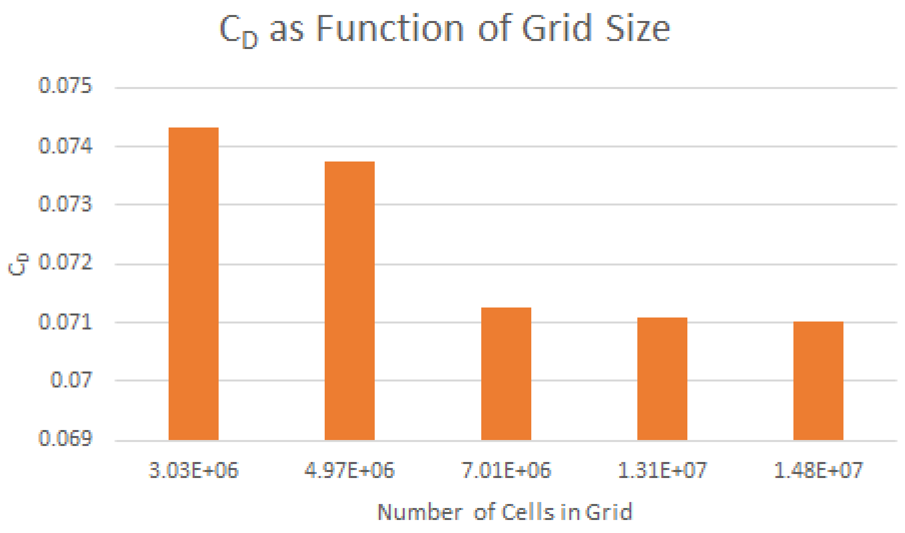

Figure 2.6: Mesh Sensitivity Graph for Determining Grid Independence

Table 2.1 summarizes the main mesh parameters used in the simulations. These parameters were used for both the baseline and modified vehicles.

Table 2.1: Main Mesh Parameters Used for Simulations

Parameter	**Value**
Base Size [mm]	13.8-14
Number of Prism Layers	12
Prism Layer Thickness [mm]	24
Volumetric Control ”near field” [% of base]	500
Volumetric Control ”mid field” [% of base]	1000
Trimmer Wake Refinement [% of base]	50
Max Cell Size [% of base]	3600

2.2.3 Physics Settings

Star CCM+ provides many options for the user to specify the type of flow in a simulation, including settings for both laminar and turbulent flow. In the real case, the Supermileage vehicle travels at a slow speed compared to regular cars and trucks, with a top speed around 20 mph. This translates to a maximum Reynolds number of 1.6×10^6. Thus it was expected that the presence of both laminar and turbulent flow would be found around the vehicle. In Star CCM+, with the turbulence model used in these simulations, the only transition models that were available to use were ones that forced transition at a user-defined location (Turbulence Suppression Model), or models that required fine tuning specific parameters with the help of experimental data (GammaReTheta model) [6]. Since the exact location of flow transition was not known, and since no experimental data was available for the specific Supermileage vehicle shape that was studied, it was decided to run simulations under laminar flow and separate simulations under turbulent flow models. The fully laminar flow simulations would provide the ”ideal” case with laminar boundary layers over all of the body, and give the lowest result for C_D. The fully turbulent case would provide turbulent boundary layers over all of the body surface, and give the highest C_D. The actual C_D would then fall somewhere in between the laminar and turbulent case since transition would occur somewhere on the rear portion of the vehicle in reality. Considering that the goal was to achieve the lowest drag area, laminar flow was chosen when making comparisons between the baseline and the other cases. Also, since laminar flow is more prone to separating, using laminar flow would help to make flow separation areas more obvious.

In order to make comparisons with the published data for PacCar II, the velocity was set in order to simulate at a Reynolds number of 1.6×10^6. This velocity was specified at the inlet and was also used as the tangential velocity of the ground floor in order to more realistically represent real driving conditions. For turbulence modeling, the k-ω turbulence model was used, which was suggested from the CD Adpaco training material and gave good convergence for both C_D and C_L. For the rest of the physics model, constant density, segregated solver, steady, 3-dimensional settings were used.

2.2.4 Monitoring Simulations

The residuals from the governing equations for the laminar and turbulent flows were monitored through all the iterations. Convergence was achieved when the residuals were no longer changing or when oscillations were sufficiently small. Most residuals dropped 1 to 3 orders of magnitude and then showed very small oscillations about one value for the rest of the iterations. For laminar flow, residuals ranged from 0.0001 to 0.1. Turbulent residuals ranged from 1.5×10^{-6} to 0.1 and usually attained a constant value. Initially, it was found that the under-relaxation factors for the segregated solver setting had to be lowered to achieve convergence. Lowering the under-relaxation factors required an increase in the number of iterations, thus upwards of 4000 iterations were specified. Along with the residuals, the coefficient of drag and lift were monitored over the iterations, and convergence was achieved when these values became constant or were showing small oscillations about one value.

2.2.5 BYU Supercomputer and Post Processing

In order to shorten the solution time, simulations were run on the BYU Supercomputer, which is part of the Fulton Supercomputing Laboratory (FSL). With a typical desktop computer, simulations would take over 10 hours to complete. The BYU FSL contains 16928 processor cores and a total memory of 50.7 TB. Each processor core is similar in computing power to a desktop, but the cores can work in parallel to run a job, and this greatly reduces computing time. The exact number of nodes dedicated to a specific simulation varied depending on the resources available, but usually 18-20 nodes were used for one job, reducing the computing time to 2.5-3 hours. To have Star CCM+ submit simulation jobs to the Supercomputer, a PBS batch script was created specifying the software and the resources for the job. The batch script contained all the necessary syntax for the Supercomputer to recognize and run a job. Multiple jobs could be submitted simultaneously, and the Supercomputer would automatically schedule the jobs as part of the overall job queue. In this way multiple simulations could be run at the same time, reducing the amount of time required for a design study. The batch script for submitting jobs also contained the name of the java file that served as the commands for Star CCM+. The java file contained all the mesh settings, physics settings, and commands for Star CCM+ to import the surface mesh of the vehicle, apply all the correct settings, and run the simulation. The java file itself was created by recording a macro in which all button clicks to produce a simulation file in Star CCM+ were recorded. The macro would automatically write each step from the user in the java code (including importing a modified surface mesh generated from Sculptor), and the code could then be called by Star CCM+ as part of the PBS batch script. The code could easily be adapted after the macro was recorded for different simulations. Appendix **??** contains examples of both the PBS batch script and the java macro used for simulations. While the simulation was iterating on the Supercomputer, the residuals could be monitored, along with any other reports that were needed, such as the coefficient of drag and lift. Figure 2.7 shows the flowchart for running jobs on the Supercomputer.

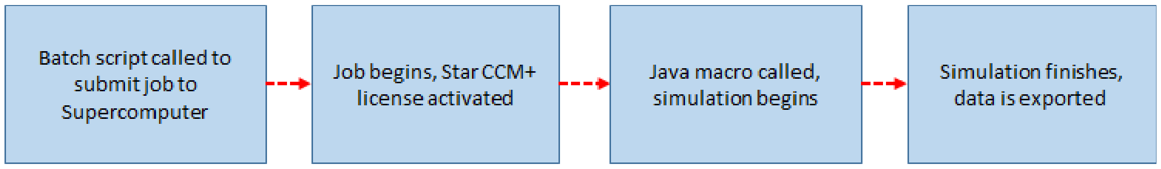
Batch script called to submit job to Supercomputer
Job begins, Star CCM+ license activated
Java macro called, simulation begins
Simulation finishes, data is exported

Figure 2.7: Process for Running Simulations on the BYU Supercomputer

Once simulations were completed, graphs of residuals were exported to check for any instability, along with the iteration history of C_D and C_L. The values for C_D and C_L were averaged over approximately the last 1000 iterations, following guidelines from CD Adapco [5]. Graphs relating to velocity and pressure fields were also exported for analysis.

2.3 Mesh Deformation

A mesh morphing program called Sculptor, developed by Optimal Solutions (see `www.gosculptor.com`), was used to alter the shape of the baseline vehicle to study different designs and their effect on aerodynamic drag. Since a full optimization was not done, changes to the current shape were driven by the designer. In other words, the design of the shape was changed by the user within physical constraints that were determined from measurements made on the vehicle. The user was free to move as close to the physical constraints as possible. In other words, the question was asked: how far can we shrink the vehicle before it becomes too small? The changes that were made to the shape of the baseline vehicle to develop a new model focused on decreasing the frontal area and creating a new shape for the front wheel cowl. Changes to individual sections such as the frontal area were applied first, resulting in a number of intermediate models that were tested (including models named A and B). Once the smaller changes were made and the performance change was noted, all the changes were brought together in a final model labelled Model C. All the changes that were made to the vehicle were based off of static measurements on the current baseline vehicle. The measurements provided limits on how far surfaces of the vehicle could be deformed. For example, measurements were made on the current width of the front wheel cowls in order determine how much space was needed for the wheels to articulate. A minimum width for the size of the front cowl was then recorded and acted as a limit for cowl shrinking in Sculptor.

2.3.1 Changing Surface and Volume Mesh

Sculptor is a mesh morphing program from Optimal Solutions that provides a method for deforming a CAD or mesh model using arbitrary volumes created by the user. The volumes, called ASD volumes (for arbitrary shape deformation), are constructed by a configuration of control points. The control points of the volume can then be moved, affecting the mesh contained in the volume. An example of an ASD volume applied to the Supermileage vehicle is shown in Figure 2.8. In this example two control points of an ASD volume are selected and ”pulled”, which stretches the front cowl. Changes to the vehicle shape in Sculptor were done on the surface mesh,

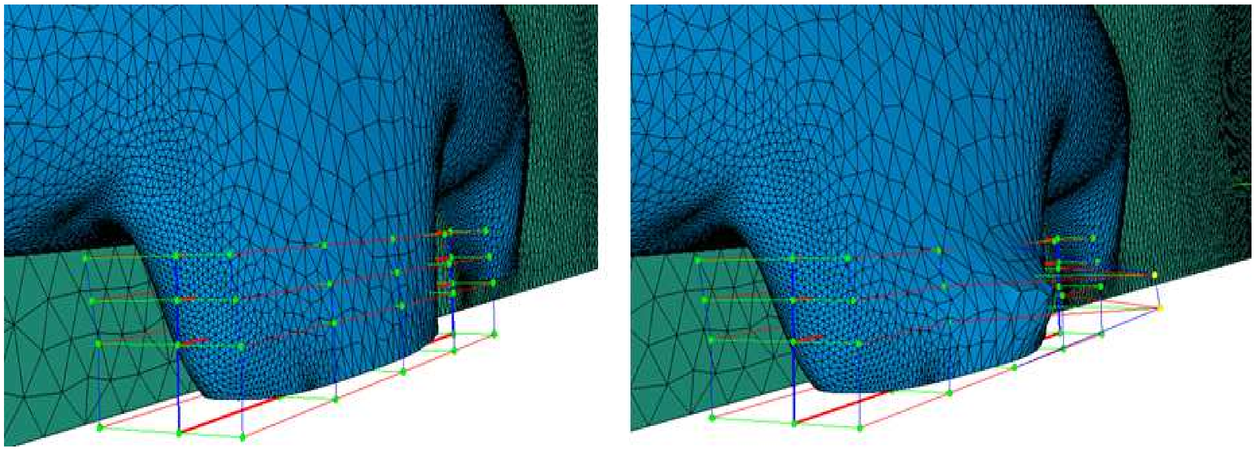
Figure 2.8: Morphing the Surface Mesh with ASD Volumes

as seen in Figure 2.8. This helped with avoiding bad quality cells, since Star CCM+ would re-mesh the volume mesh on top of the surface mesh changed in Sculptor. The surface repair tool in Star CCM+ was also used to make sure a high cell quality was maintained.

2.3.2 Design of Experiments (DOE) using Sculptor

The optimization package in Sculptor allowed for more in-depth study of different vehicle shapes and their effect on C_D by creating a number of ASD volumes and assigning control points as analysis variables. These analysis variables could then be changed randomly within maximum and minimum limits set by the user, and the resulting geometries could be imported into Star CCM+ for testing. With this method of random perturbations (called Optimal Latin Hypercube), many different shapes could easily be generated and tested. The maximum and minimum limits were determined from measurements made on the current vehicle. Section 2.8 provides an example of using the Opimal Latin Hypercube method to study a number of front cowl designs. While not a full optimization study, the DOE provided opportunities to evaluate a number of designs that had not been considered previously.

2.4 Model C and Physical Constraints

The following sections explain all the changes that were made to the baseline vehicle to develop Model C. As mentioned previously, the driving force behind the design changes was the desire to shrink the vehicle as much as possible, with each change in design driven by the designer. However, there were physical constraints on the the amount of shrinking that could be done before the vehicle became too small for the driver or the interior components. This was especially true for the wheel cowls, since there needed to be room for the wheels to fit into the cowl while still maintaining a small shape. The main cowl constraints are summarized in Table 2.2. In addition to the cowl constraints, additional constraints were considered for the vehicle thickness at multiple locations, and the reduction of the roof height. For example, the minimum vehicle thickness at the rear of the vehicle varied with height in order to save room for the engine and rear wheel components. At the height where the wheel axle was located, a minimum thickness of 7.5 inches was needed. For the top of the vehicle, it was determined that the roof could be lowered by 1 inch everywhere while still leaving room for the driver and firewall installation. These constraints were determined by taking measurements on the current baseline vehicle.

Table 2.2: Important Physical Constraints on Wheel Cowl Shape Changes

Vehicle Part	**Minimum Size [inches]**
Bottom of Front Cowl- width	3
Bottom of Front Cowl- length	19
Front Cowl length near vehicle floor	5
Bottom of Rear Cowl- width	1.5

2.4.1 Decreased Frontal Area

One of the most effective ways to decrease the drag area of the vehicle was to decrease the frontal area. From measurements on the baseline vehicle, it was determined that there was space to lower the top of the vehicle without compromising the driver or the firewall. This was done by lowering the top of the vehicle by 1 inch. Figure 2.9 illustrates the parts of the vehicle that were lowered.

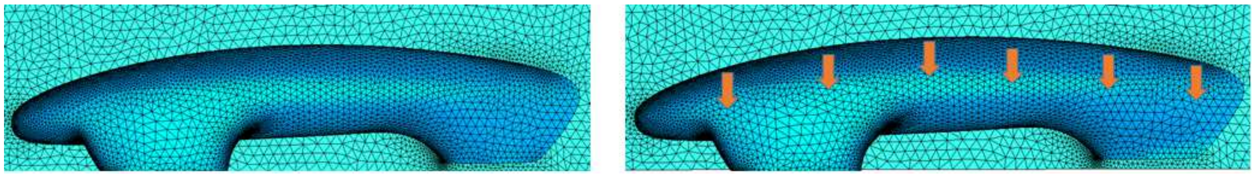

Figure 2.9: Baseline vehicle (left) and Model C (right) with Lowered Height

2.4.2 Smaller Front Cowl

One of the biggest changes for Model C was the front cowl shape. The PacCar II engineers noted that an asymmetric airfoil shape for the front cowl might help reduce flow separation at the back end of the cowl and could also help reduce the Venturi effect in the underbody of the vehicle [1]. The Venturi effect describes the decrease in pressure that results from the increase in velocity in a section with decreasing cross sectional area, such as around the front cowls. This pressure decrease causes a pressure difference across the upperbody and underbody of the vehicle, resulting in a larger downforce and a higher rolling resistance, which is undesired.

With these considerations, the front cowl was shrunk 2.3 inches (at its thickest point) to an asymmetric shape while still allowing room for the front wheels to turn. Figure 2.10 shows the comparison between the original and modified cowl shape. The length of the cowl was also shortened 1 inch in an effort to reduce the wetted area, with the aim of reducing the skin friction drag.

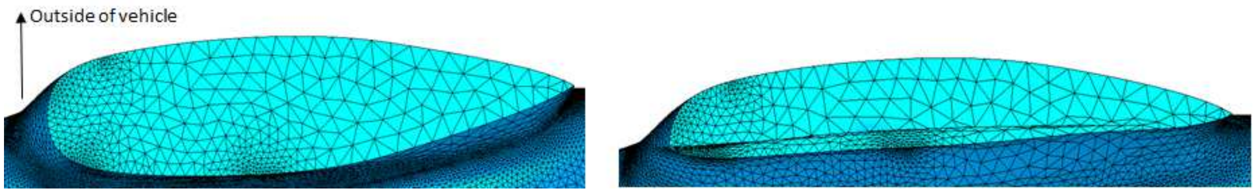

Figure 2.10: Baseline vehicle (left) and Model C (right) with Asymmetric Cowl

2.4.3 Smaller Rear Cowl

Measurements on the current vehicle also indicated that there was space to reduce the size of the rear cowl. Since the rear wheel doesn't turn, there was more freedom to shrink the cowl to a small size. The constraint for shrinking the bottom of the rear cowl was the thickness of the rear tire. The constraints for shrinking the rest of the cowl above the bottom were related to leaving room for the axle and chain and engine components. Thus the rear cowl could only be shrunk up to a certain height from the bottom of the vehicle. Shrinking the rear cowl also contributed to a smaller frontal area, since a projection of the shape of the vehicle from the front includes the rear cowl, as seen in Figure 2.12. Figure 2.11 shows the comparison for the rear cowls. The bottom of the rear cowl was shrunk 2.5 inches. Since the plane of symmetry runs down the middle of the vehicle, only half of the rear cowl can be seen in Figure 2.11.

A comparison of the front of the vehicle is shown in Figure 2.12. The green section is the symmetry plane in the domain, while the blue section represents the mesh of the vehicle. The

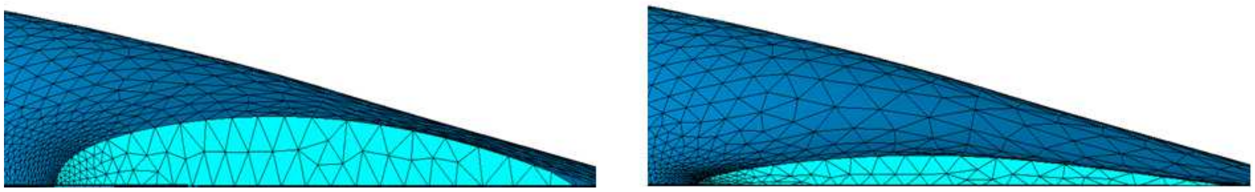

Figure 2.11: Baseline vehicle (left) and Model C (right) with Smaller Rear Cowl

changes to the front cowl are the most evident. The rear cowl change can also be seen in the figure.

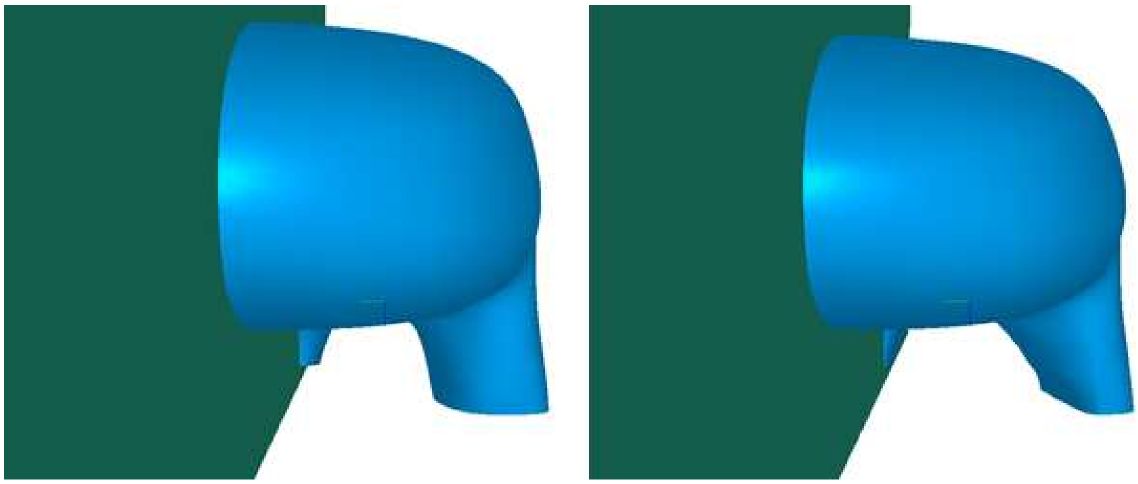

Figure 2.12: Comparison of Frontal Areas of Vehicle (Left = Baseline, Right = Model C)

2.5 Validation of Simulation Model Settings

2.5.1 Sphere and Circular Cylinder Simulation

Test cases were done on a circular cylinder and sphere. This was done in in order to gain experience with using Star CCM+, and to show that C_D and flow separation could be predicted under laminar flow and also turbulent models. Since there is already published data available for drag and flow over a circular cylinder and sphere, it is possible to compare the CFD results to experimental data to confirm that the simulations (with the same settings as used on the vehicle) are accurately modeling the flow. The same turbulence model used on the vehicle was used for the test cases, as well as the same volume mesher. The diameter of the sphere and cylinder was made the same size as the length of the vehicle. For laminar flow, the sphere case was run at a Reynolds number of 1000. Figure 2.13 shows a picture of the volume mesh for the sphere, with a symmetry plane cutting the model in half as in the vehicle simulations. The value obtained for C_D at Re = 1000 was 0.58, which is in good agreement with Fox [7] who reports a value around 0.59, and Hoerner [8] who shows approximately the same value. The prism layers and total number of cells were changed from the vehicle values in order to optimize the mesh for the simulation.

For the circular cylinder case, simulations were run under laminar flow, with additional simulations done under k-ω turbulence models. Laminar flow was run at a Reynolds number of 1000, and the simulations with turbulent modeling was done at a Reynolds number of 1.6×10^6 in order to match the vehicle simulations. The laminar C_D value obtained was 0.9, which falls about 10% within the experimental value of 1 reported in Anderson [9]. For laminar flow, experiments show that separation should occur about 80 degrees from the stagnation point [9]. Figure 2.13 shows the velocity plot over the circular cylinder. Separated flow can be seen just before the vertical, showing good agreement with 80 degree separation. The C_D obtained for turbulent flow was 0.35, which also shows good agreement with experimental values from Anderson, which range from 0.29 to about 0.37 [9]. These test cases show that using similar parameters to the ones used on the supermileage vehicle give accurate results for the circular cylinder and sphere, giving confidence in using the C_D values generated from the vehicle CFD simulations.

2.5.2 Separation over Vehicle

More test cases were done by showing that separation could occur with a model of the baseline vehicle. Figure 2.14 shows the velocity magnitude plot along with velocity vectors for a modified vehicle shape. A section of the front nose was raised in an exaggerated manner in order to force separation. Flow reversal can be seen behind the raised section, which gives confidence in observing flow separation for the regular baseline and Model C test cases. In Figure 2.14, the number of cells in the mesh was 1.1×10^6, laminar flow was simulated, and a trimmer wake refinement was used as in the baseline and Model C test cases.

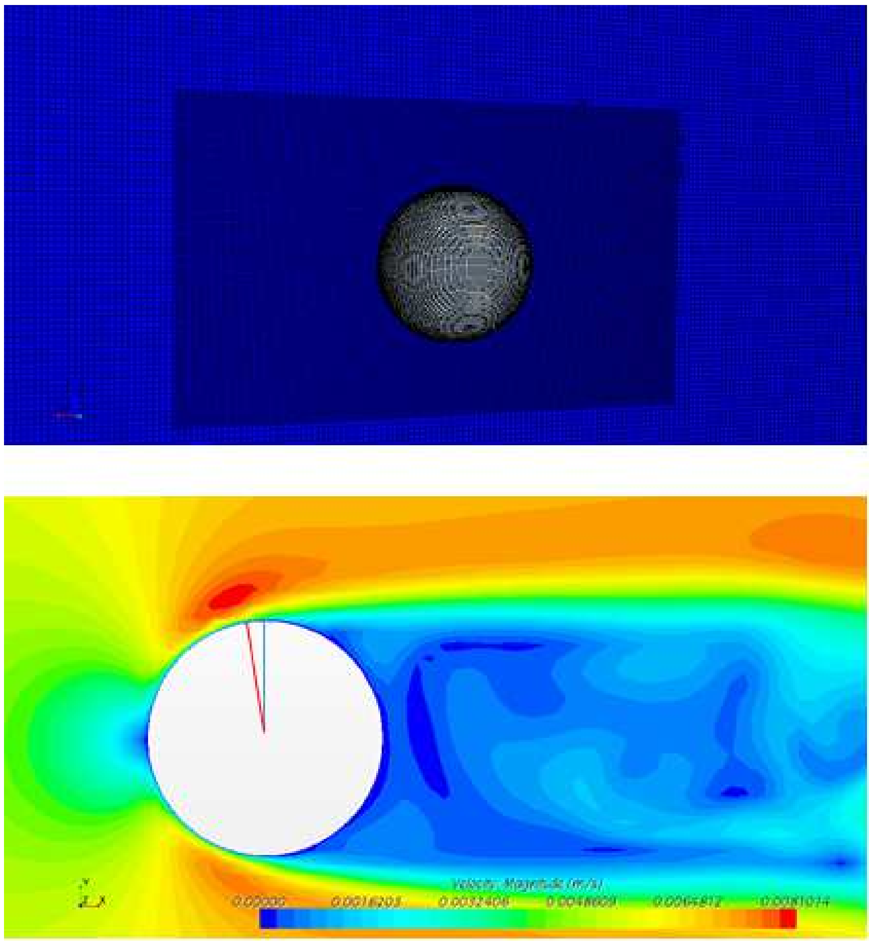

Figure 2.13: Sphere Volume Mesh (top) and Circular Cylinder Velocity Plot (bottom) with Red Line showing Approximate Separation Point

2.6 Error Analysis in CFD Simulations

There are a few sources that can contribute to the error in a CFD analysis [10]. Some sources include modeling approximation error, computer round-off error, and discretization error. Discretization error arises from representing the governing flow equations in a discretized spatial domain. This error can be minimized with a grid convergence study (as mentioned in section 2.2.2). It can also be quantified with a method called Grid Convergence Index (GCI), which is based off of Richardson Extrapolation (explained in [11] and [12]). In order to use GCI, different simulations containing successively finer grid sizes were run and the values of C_D were obtained. Three simulations were run on the baseline vehicle, with 2×10^6, 4×10^6, and 8×10^6 cells (labelled f_3, f_2, and f_1, respectively). Two GCI values are obtained; GCI_{12} for the finer mesh and

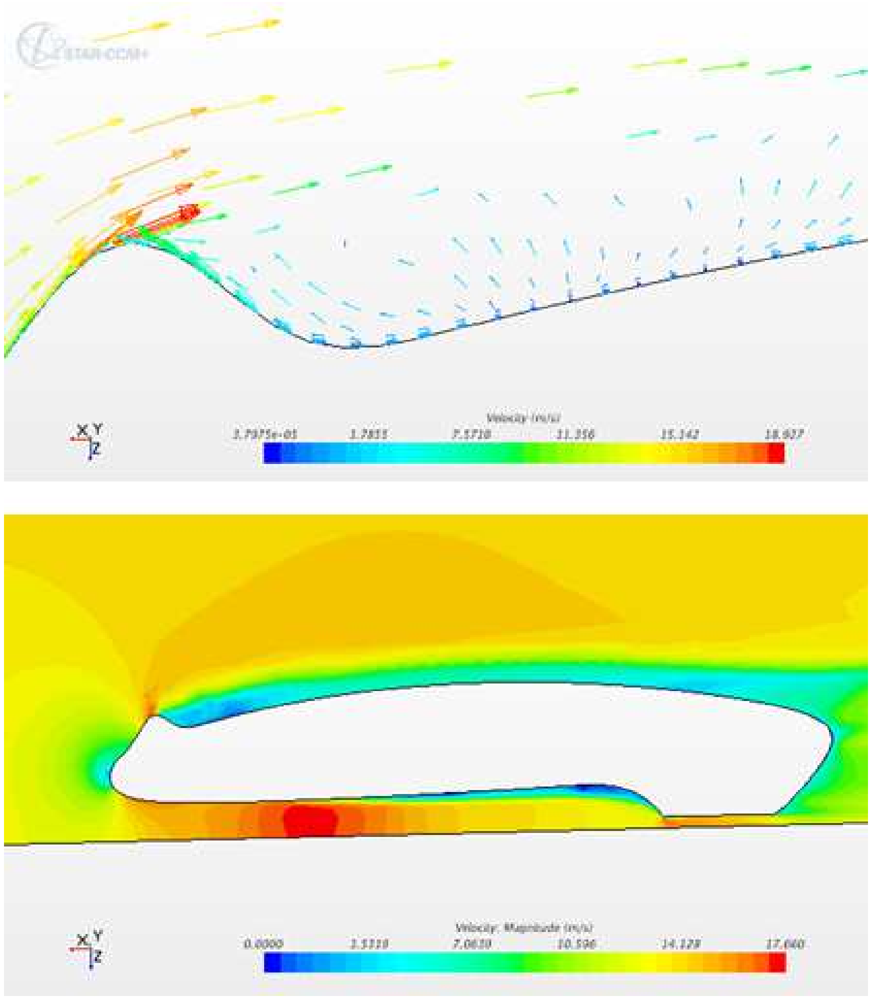

Figure 2.14: Velocity Vectors (top) and Magnitude on Symmetry Plane (bottom) for Vehicle Flow Separation Test Case

GCI_{23} for the coarser mesh. GCI_{12} was used to quantify the discretization error for the vehicle simulations since the mesh for f_1 was closest to the actual mesh size used. From the GCI calculations, it was found that the error band is 3.5%, which can be applied to the C_D results from the CFD simulations. For example, the laminar C_D for the baseline vehicle is $0.069 \pm 3.5\%$ or 0.069 ± 0.002. For the turbulent case, $GCI_{12} = 0.007$, thus the turbulent C_D for the baseline vehicle can be stated as $0.107 \pm 0.7\%$ or $0.107 \pm 7.5 \times 10^{-4}$. Appendix A contains the steps for calculating GCI.

2.7 Results

Table 2.3 shows the laminar drag results from the simulations run on the baseline vehicle and Model C. Comparison is also made with the published data on PacCar II [1]. As mentioned in section 2.3, around five to six intermediate models were tested before arriving at the final shape for Model C. Since Model C incorporated all of the shape changes that were possible, this model was chosen for comparison to the baseline vehicle and PacCar II. The error bands from the GCI calculations are also included in the table. As Table 2.3 shows, the drag area for Model C is reduced by almost 17.4% when compared to the baseline vehicle. This reduction comes from a reduced frontal area (almost 7%) and reduced C_D (almost 11%). The coefficient of lift was also reduced by about 18% (from $C_{L,Baseline} = -0.367$ to $C_{L,ModelC} = -0.30$). Both the C_L and C_D reductions have an effect on the fuel efficiency of the vehicle, which will be discussed in Chapter 4. In order to determine the effect of the new Model C shape on the surrounding flow field, velocity and pressure plots were generated and analyzed.

Table 2.3: Laminar Drag Results from Simulations Compared with Published Data

Model	C_D	**Frontal Area (m^2)**	**Drag Area (m^2)**	**C_D % Change from Baseline**	**Drag Area % Change from Baseline**
Baseline	0.069 ± 0.002	0.337	$0.023 \pm 8.05 \times 10^{-4}$	-	-
Model C	0.061 ± 0.002	0.314	$0.019 \pm 6.65 \times 10^{-4}$	-10.77	-17.39
PacCar II	0.07	0.254	0.018	1.43	-21.74

Table 2.4 shows the fully turbulent drag values for the baseline vehicle and Model C (same frontal area used in both turbulent and laminar cases), along with the error bands from the GCI calculations. The turbulent drag values are at least 1.5 times greater than the laminar values, which reveals the importance of achieving laminar flow over as much of the vehicle as possible. Chapter 4 also discusses the impact of the turbulent drag values on the fuel efficiency.

2.7.1 Velocity plots

The velocity plot on the symmetry plane of the vehicle is shown in Figure 2.15. As can be seen, the velocity in the underbody section near the front cowl is reduced from 10.87 m/s to 10.36

Table 2.4: Turbulent Drag Results from Simulations

Model	C_D	**Drag Area** (m^2)	C_D **% Change from Baseline**	**Drag Area % Change from Baseline**
Baseline	$0.107 \pm 7.5 \times 10^{-4}$	$0.0360 \pm 2.5 \times 10^{-4}$	-	-
Model C	$0.103 \pm 7.2 \times 10^{-4}$	$0.0323 \pm 2.3 \times 10^{-4}$	-3.74	-10.50

m/s. In general, lower velocities are found throughout the underbody between the cowls for Model C. The lower velocities in the Model C plot indicate a reduction in the Venturi effect mentioned previously. This helps to lower the rolling resistance (reflected in the decreased value of C_L), which is an obstacle to increasing fuel efficiency. The same velocity scale is used in each plot for easier comparison between models.

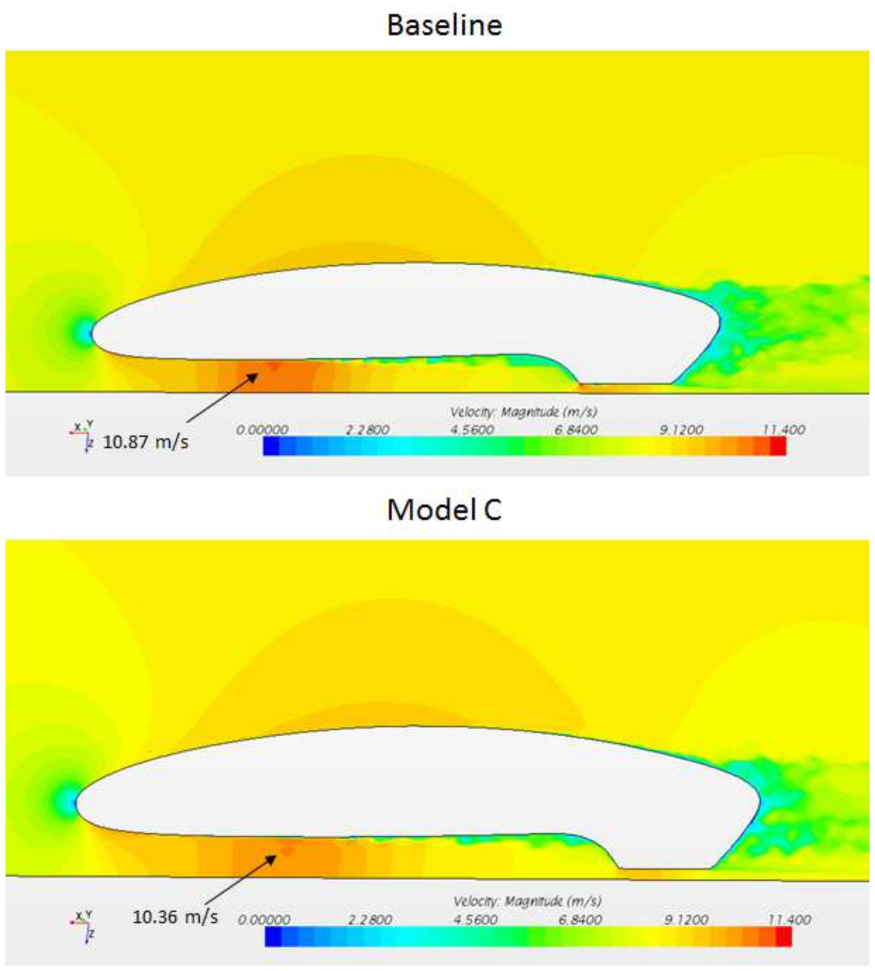

Figure 2.15: Velocity Plot of Baseline vehicle and Model C

2.7.2 Velocity around Front Cowl

Figure 2.16 shows the x component of velocity around the front cowl area for the baseline and Model C. Since the flow direction in the simulations was in the negative x direction, positive velocities in Figure 2.16 are in red and represent reversed or separated flow. The green and blue regions show flow in the freestream direction (-x). The freestream velocity is -8.61 m/s. The thinner, asymmetric shape for the Model C front cowl caused a decrease in the flow velocity. For example, in the underbody section between the cowls, the velocity was reduced from -10.7 to -10.3 m/s. In the wake region behind the cowls, higher velocities are recovered with the Model C cowl, as shown by the two locations in the figure (-6.6 versus -5.7 m/s). With higher velocities recovered behind the cowl with Model C, the momentum loss is decreased, leading to a smaller wake region, and a smaller pressure drag overall.

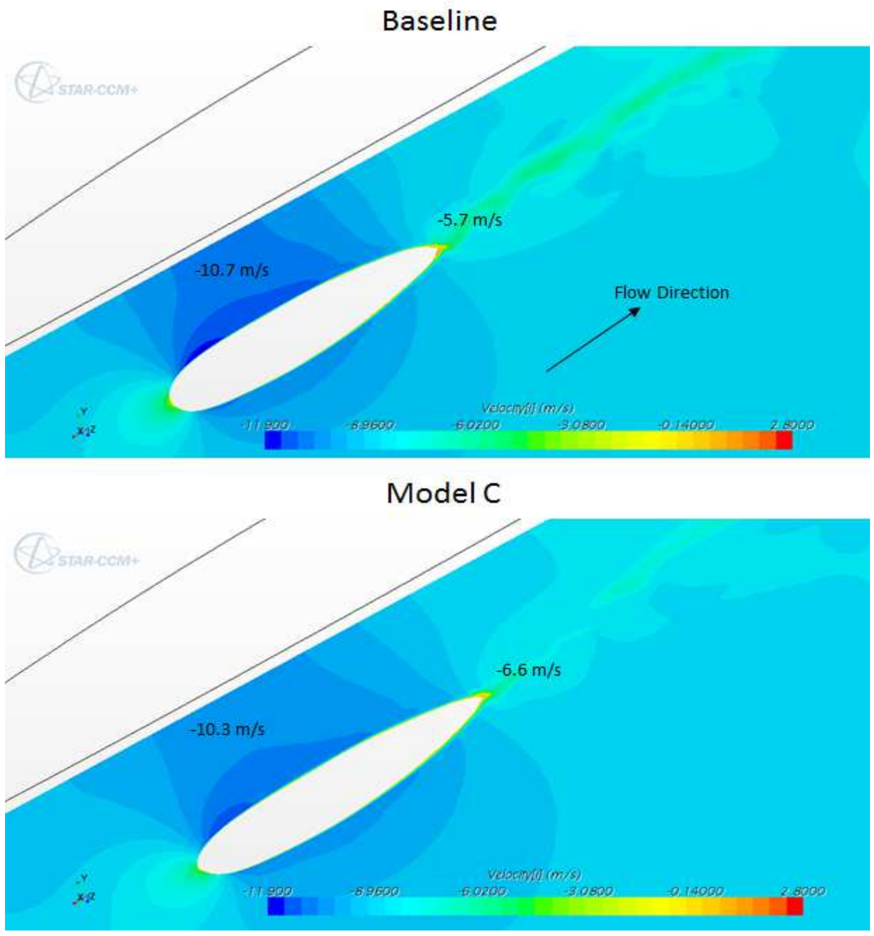

Figure 2.16: X Velocity around Front Cowl for Baseline and Model C

Figure 2.17 shows a zoomed in plot of the x component of velocity around the front cowl. Both plots show regions of dead or reversed flow behind the cowl, but the magnitude of the velocity in these regions is decreased for the Model C cowl. Reducing the extent of flow separation helps to lower the pressure drag for Model C.

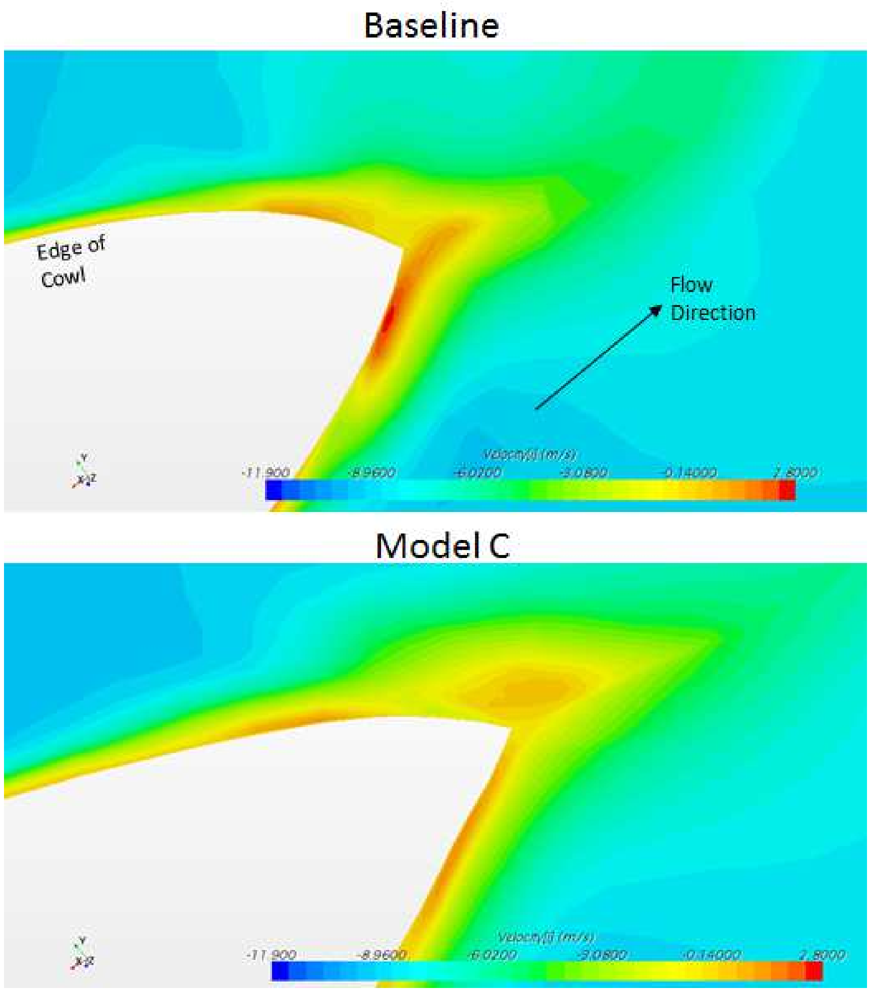

Figure 2.17: X Velocity around Rear of Front Cowl for Baseline and Model C

2.7.3 Front Cowl Pressure

Figure 2.18 shows the pressure field near the front cowl for the baseline vehicle and Model C. The ambient gauge pressure in the simulations is 0 Pa, and is set relative to atmospheric pressure. Thus negative pressures in the plot are below typical atmospheric pressures. The reduction in the

Venturi effect can be seen in these plots since the Model C pressures are higher then the baseline. For example, at the point where the lowest pressures are seen in both plots (near the front of the cowl), the pressure is increased from -40 Pa for the baseline to -36 Pa for Model C. This pressure increase in the underbody area helps to reduce the total change in pressure between the underbody and upperbody of the vehicle, which leads to a lower downforce for Model C.

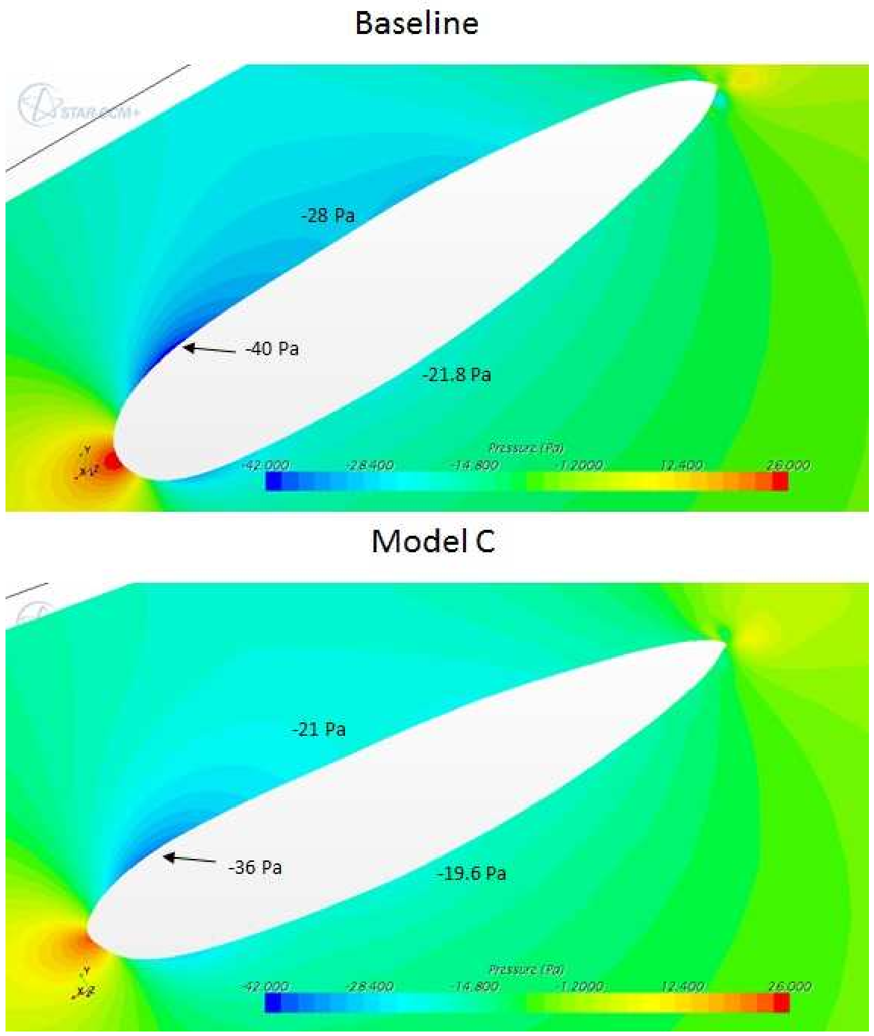

Figure 2.18: Pressure Comparison between Baseline and Model C near Front Cowl

2.7.4 Velocity Around Rear Cowl

Figure 2.19 shows the x component of velocity behind the front cowl and around the rear cowl at the back of the vehicle. Specific points are labelled with the velocity values to facilitate comparison. In the wake region behind the front cowl (bottom left in the figure), the velocity change from the freestream is smaller for Model C than the baseline. This indicates a smaller pressure difference between the upperbody and underbody for Model C in the region behind the front cowl, and also a decrease in the momentum loss. Near the rear cowl, the Model C velocity deficit is also smaller than the baseline velocity deficit (-9.8 m/s for Model C versus -10.3 m/s for baseline), which contributes to reduced shear drag on the surface of the cowl. The smaller velocity deficit, together with the reduced momentum loss and decreased shear, indicate a decrease in aerodynamic drag.

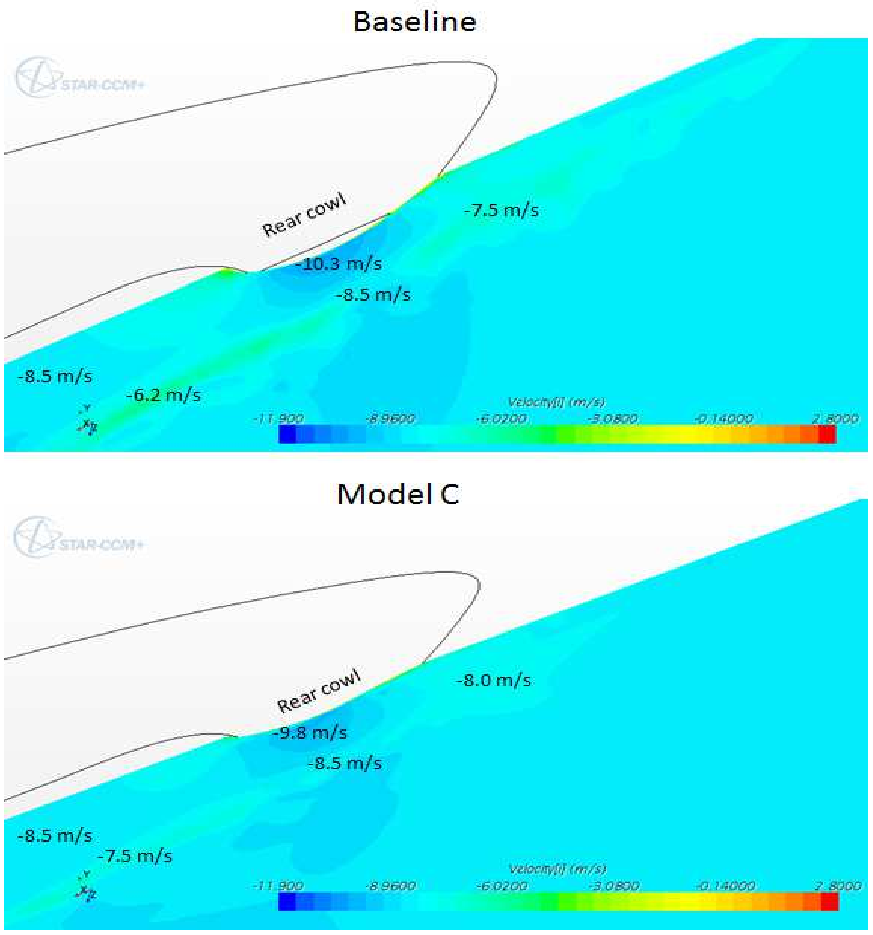

Figure 2.19: X Component of Velocity behind Front Cowl and Around Rear Cowl

2.7.5 Rear Cowl and Wake

Further effects of the rear cowl shape modification can be seen in surface pressure plots. Figure 2.20 shows the surface pressure distribution on the rear portion of the vehicle for the baseline and Model C. The thinner rear cowl on Model C caused a decrease in pressure at the very rear of the vehicle (highlighted with the black circles in Figure 2.20), from 11.89 Pa to 7.6 Pa. Increased pressures are also found on the surface of the cowl where the modifications were made. The pressure in this section is increased from -22.3 Pa to -13.8 Pa. These pressure changes indicate a change in velocity near the rear cowl, which is supported by the velocity plot in Figure 2.19.

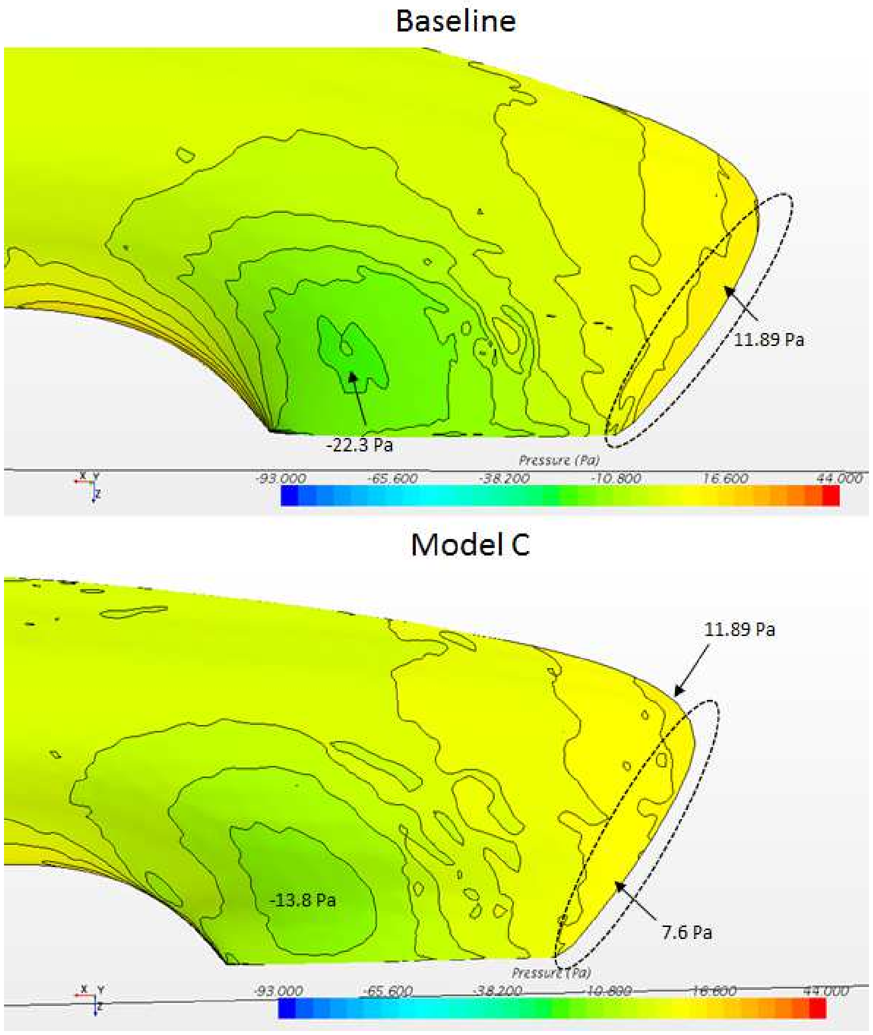

Figure 2.20: Surface Pressure on Baseline and Model C near Rear Cowl

The effect of the thinner rear cowl on the wake behind the vehicle can be seen in Figure 2.21. The wake directly behind the portion of the rear cowl that was shrunk is highlighted in black circles in the figure. In the highlighted portion, the section of low velocity (blue color in plot) extends, on average, 0.03 m behind the edge for the baseline. For Model C, this distance is shrunk to 0.018 m. The shrinking of the low velocity areas contributes to the decrease in momentum loss behind the vehicle, which contributes to a lower pressure drag.

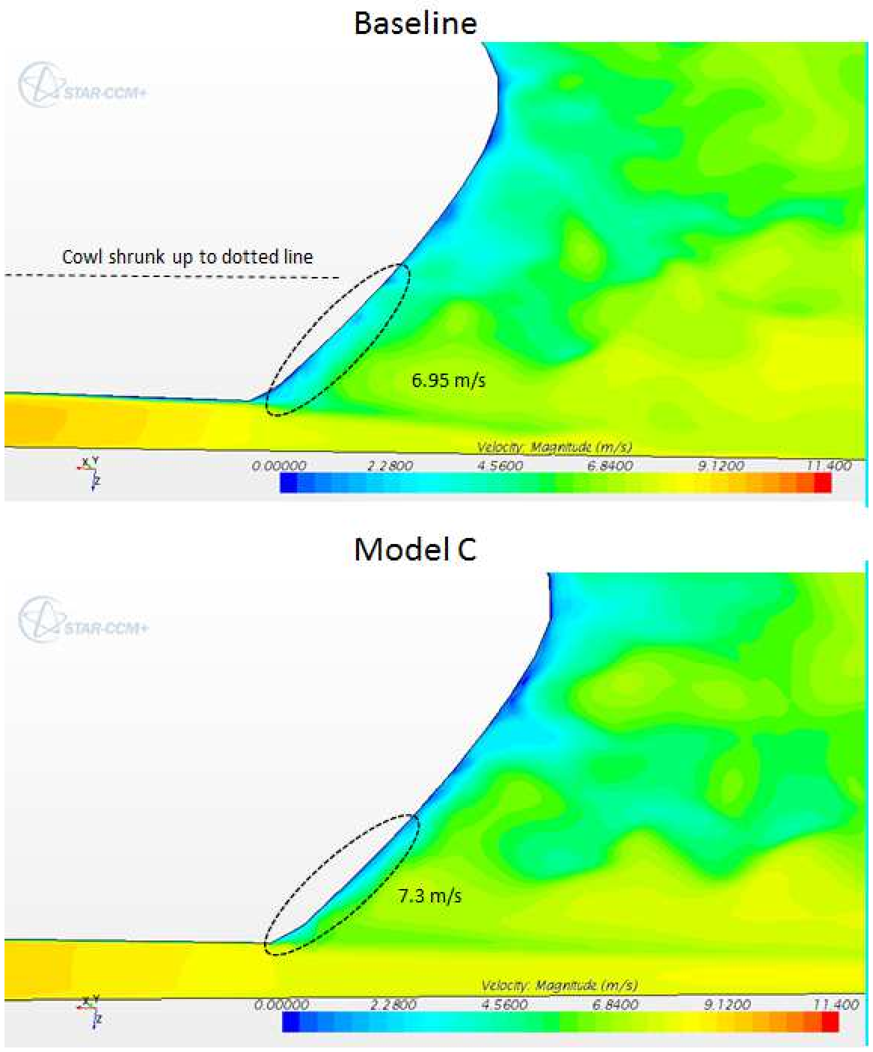

Figure 2.21: Wake Area Behind Vehicle for Baseline and Model C

2.7.6 Pressure Coefficient on Symmetry Plane

Figure 2.22 shows the coefficient of pressure (C_p) along the symmetry line of the vehicle for both the baseline and Model C. This type of plot gives an indication of the change of C_L since the pressures on the upperbody and the underbody can be compared in one graph. A smaller difference between the upperbody and underbody pressures indicates a lower downforce in general. This graph also gives an indication of the locations where the velocity was changed along the vehicle. Locations where there are differences in pressure are noted with black arrows on the graph. On the C_p plot, the front of the vehicle is located near $x = -3.5m$ and the rear is near $x = -0.5m$. Along the top of the vehicle, the pressures are very nearly equal. However the Model C pressure is changed in a few places on the underbody, specifically near x = -1.25 m (just past the rear cowl), near x = -1.6 m (the front part of the rear cowl), and x = -2.75 m (in the underbody area between the front cowls). The C_p values at these locations are in agreement with the plots shown earlier, which leads to the conclusion that the Model C shape effectively decreased the change in pressure between the upperbody and underbody , which lead to a lower C_L value for Model C (shown in the top right of the figure).

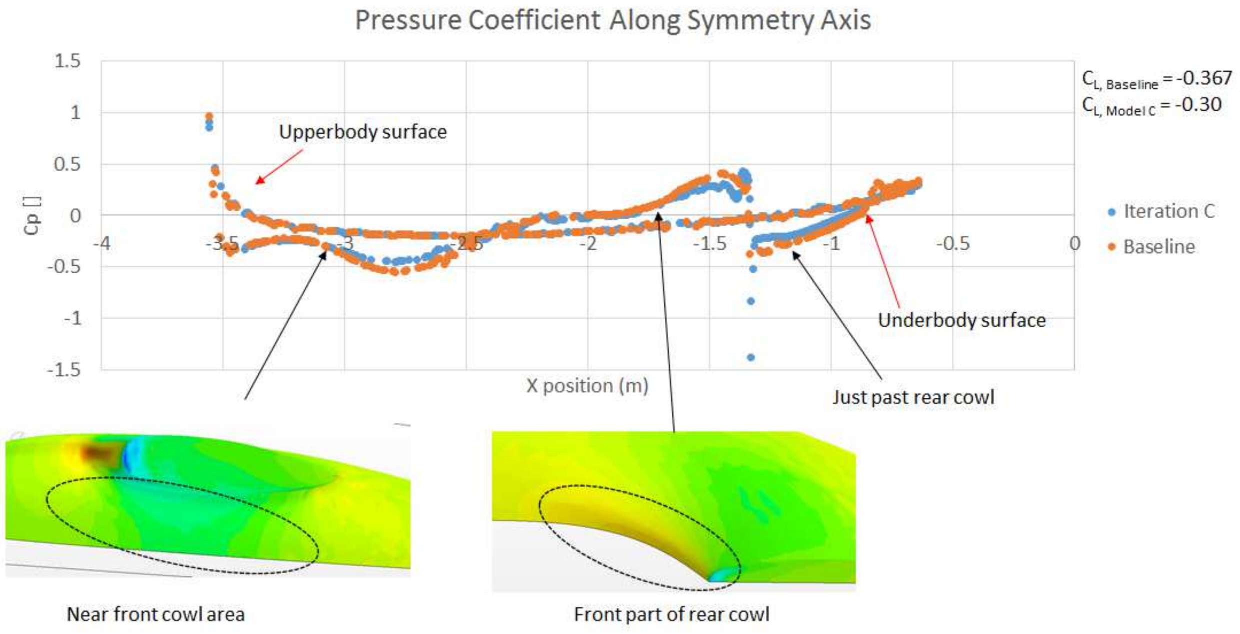

Figure 2.22: C_p Along Symmetry Plane for Baseline and Model C, with Pressure Plot Inserts Highlighting Affected Areas

2.7.7 Wall Shear Stress on Front Cowl

Wall shear stress on the body of the vehicle is an indication of the extant of the skin friction drag that is acting on the vehicle. Figure 2.23 shows the wall shear stress on the inside part of the front cowl for the baseline vehicle and Model C. The major area of wall shear stress reduction is noted in the black circles in the figure. The Model C front cowl shows reduced wall shear towards the rear. The maximum shear stress in the circle in the baseline plot is 0.485 Pa compared to 0.355 Pa for Model C. The reduction in shear stress also helps to reduce separation directly behind the cowl as noted in Figure 2.17. A small area of increased wall shear is seen on the front of the Model C cowl due to the skinnier frontal profile. A discussion of the overall friction drag in comparison with the pressure drag is given in section 2.7.8

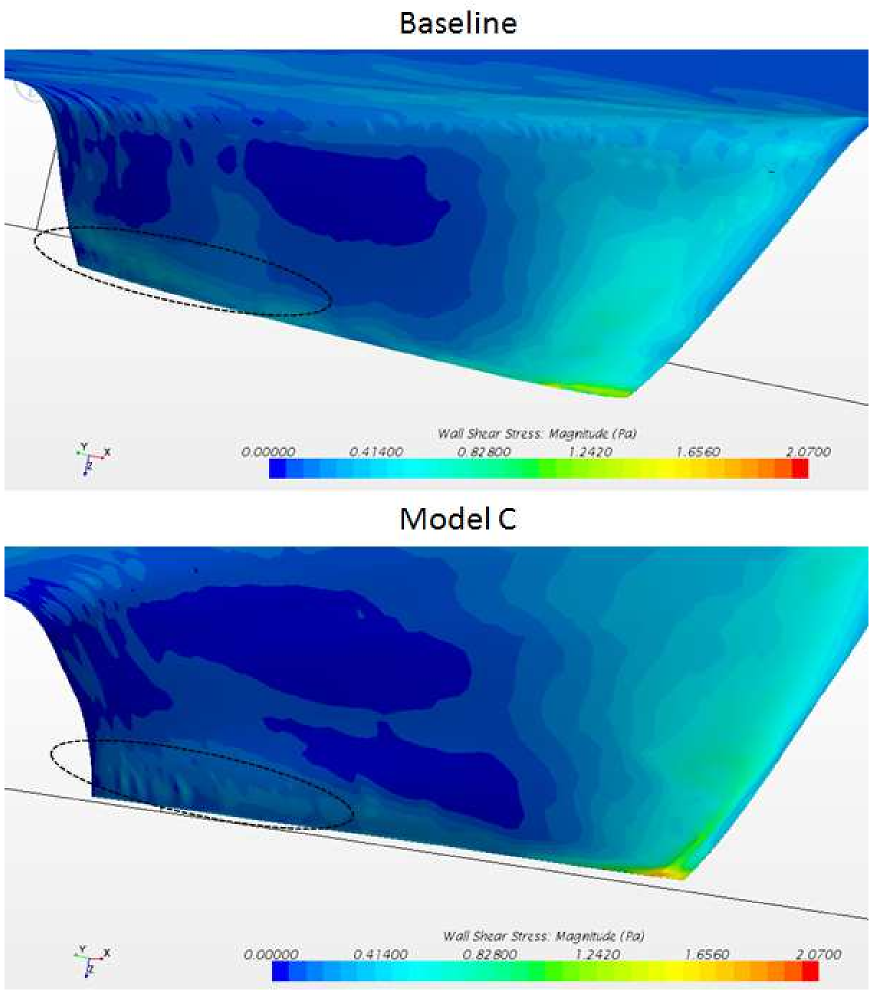

Figure 2.23: Wall Shear Stress Comparison on the inside of the Front Cowl

2.7.8 Breakdown of Drag Components

Star CCM+ allows the user to view the contribution of the pressure and shear drag to the overall drag for further study. Table 2.5 shows the corresponding contribution for both fully laminar and fully turbulent flow. The table shows that under laminar flow, the pressure/shear ratio moves closer to 50/50. Under turbulent flow, the ratio moves further away from 50/50, with the shear drag being the most dominant. Under both laminar and turbulent flow, the pressure drag was reduced for the Model C shape.

Table 2.5: Shear and Pressure Drag for Baseline and Model C as Percentage of Total Drag

	Baseline [ave C_D value]	**Model C [ave C_D value]**
Laminar		
Shear	40% [0.0277]	47% [0.0289]
Pressure	60% [0.0404]	53% [0.0322]
Turbulent		
Shear	61% [0.0656]	66% [0.0686]
Pressure	39% [0.0419]	34% [0.0346]

In relation to Table 2.5, Figure 2.24 shows the pressure and shear drag for a streamlined strut based on its thickness to chord ratio from Fox [7]. The corresponding drag *distributions* (not the actual drag values) for baseline and Model C are placed on top of the graph (BL for baseline laminar, CL for Model C laminar, BT for baseline turbulent, and CT for Model C turbulent). The graph shows minimum drag is achieved with a pressure/shear ratio that is close to 50/50, thus it is desirable to achieve the Model C laminar drag ratio since its drag ratio is the closest to 50/50. The turbulent drag ratios move further away from the minimum drag point, which supports the conclusion from the CFD results that turbulent flow is undesired. The current vehicle thickness to chord ratio is 0.26, which corresponds very nearly to the minimum drag value on the graph. The exact values from the graph cannot be applied to the vehicle since the results are for 2D flow over the strut, and hence no ground effects are considered. However, the graph does provide a good standard so that the baseline and Model C's drag distribution can be compared.

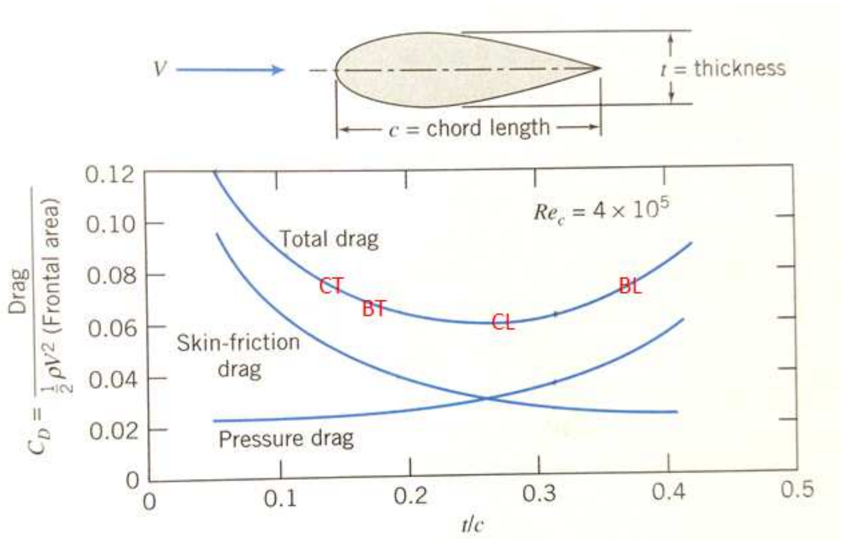

Figure 2.24: C_D for Streamlined Strut from [7]

2.8 Design of Experiments (DOE) for front cowl

The Optimal Latin Hypercube function within the Optimization Center in Sculptor was used to generate seven additional shapes for the front wheel cowl. This was done in order to investigate if there were thicker cowls that had the shape as the Model C cowls that could provide lower drag. Seven shapes were generated all within the constraints provided by measurements on

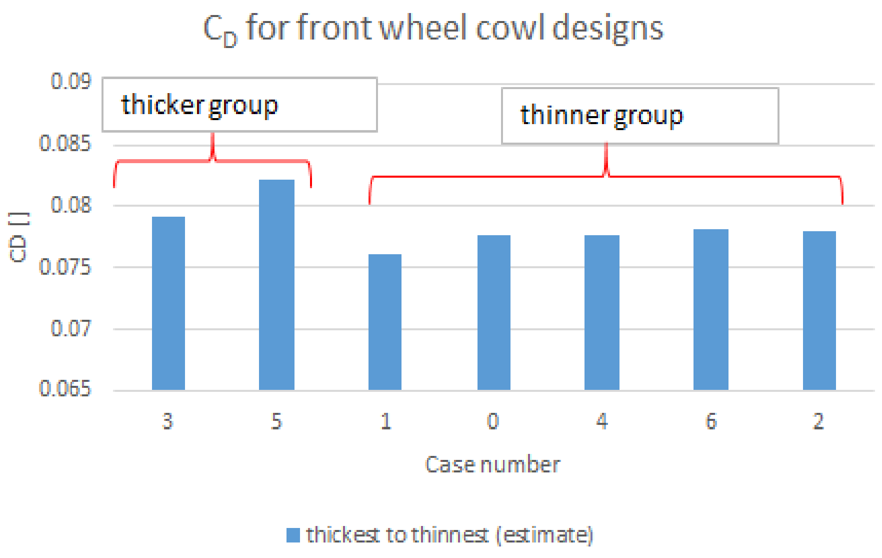

Figure 2.25: C_D for Front Cowl Designs Generated from Sculptor

the current vehicle. The designs were randomly generated, then grouped into a thicker group and a thinner group, depending on the thickness of the front cowl. None of the wheel cowls generated were thinner than the Model C front cowl since the Model C cowl represented the thinnest possible cowl that could be shaped without breaking the constraints. Figure 2.25 shows the coefficient of drag for the different designs. As expected, the thinner wheel cowls provide the lower values for coefficient of drag, which gives confidence in using the thinnest wheel cowl possible for the front wheel. This study, while not a full optimization, helped in the decision to use Model C cowls for the new vehicle design.

CHAPTER 3. EXPERIMENTAL WORK

3.1 Overview

The experimental portion of this thesis work consisted of wind tunnel testing of scale models of the baseline vehicle, Model C, and PacCar II. This was done in order to validate the findings from the numerical simulations; mainly that the Model C shape results in lower drag area when compared to the baseline.

3.2 3D Prints of the Vehicles

Stereolithography (STL) files of each of the models were prepared either from the original CAD model or the surface mesh representing the entire vehicle. These STL files were used for making the 3D prints. The models were printed in the Precision Machining Lab in the Crabtree building on BYU campus. A Stratasys uPrint 3D printer was used. The printer had a 6”x6”x8” printing section and could print ABS plastic parts with soluble support. The STL models were scaled down from the full size to fit on the diagonal of the printing bed. The resulting length of the parts was about 10 inches, giving a scale of 1:11.5, and each part took about 6 hours to complete, including dissolving of the support material. Figure 3.1 shows some examples of prints from the Stratasys uPrint 3D printer.

An AirWolf 3D printer in the basement of the Clyde building on BYU campus was also used to make practice parts as needed. This printer had a 12”x8”x10” printing section and could print ABS and PLA plastic. This printer was used to make smaller scale models in order to practice vapor treatments on the surface, and also to develop wind tunnel attachment ideas. Figure 3.2 shows a picture of the AirWolf 3D printer and the printing process. Since the support material could not be removed without leaving extra material on the underbody surface, these parts were

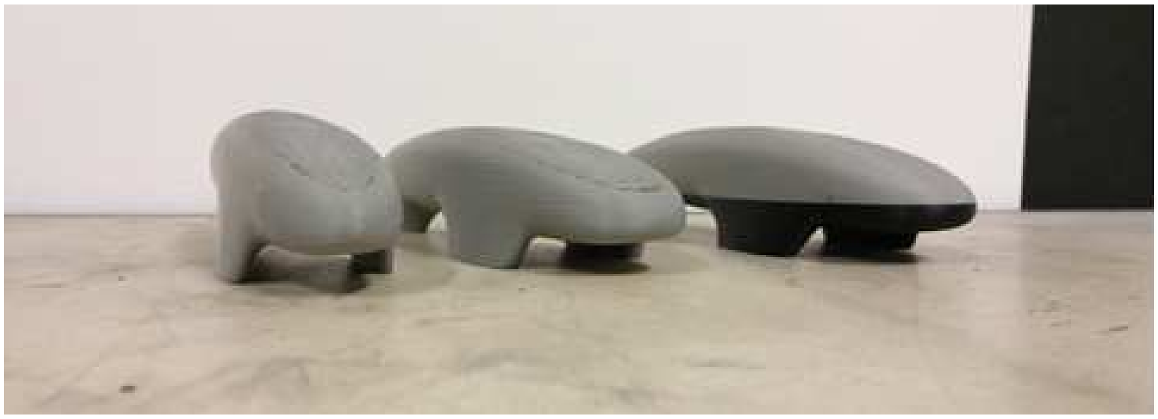

Figure 3.1: Examples of 3D Prints

not used for wind tunnel testing. Parts from this printer took anywhere from 4 to 8 hours to complete printing, depending on the model size.

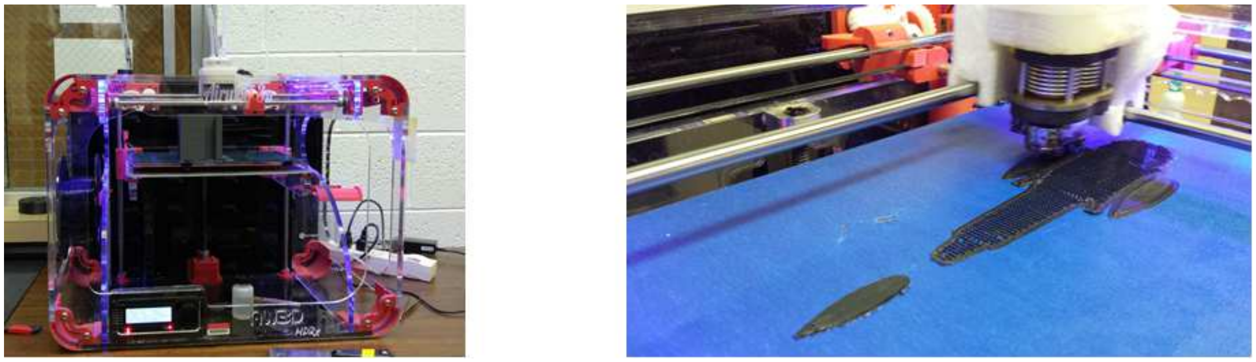

Figure 3.2: AirWolf 3D Printer (left) and Photo taken during Printing Process (right)

3.3 Wind Tunnel

The wind tunnel in the basement of the Clyde building on BYU campus was used for the experimental work. Figure 3.3 shows a photo of the wind tunnel. The wind tunnel is open type with test section dimensions of 1.5 $\times$ 1.5 ft, and a maximum speed of 110 mph. With the scale models in the test section, the blockage ratio φ was calculated to be 1.16 - 1.18%, which falls well within the blockage ratio limits for typical automotive testing [2].

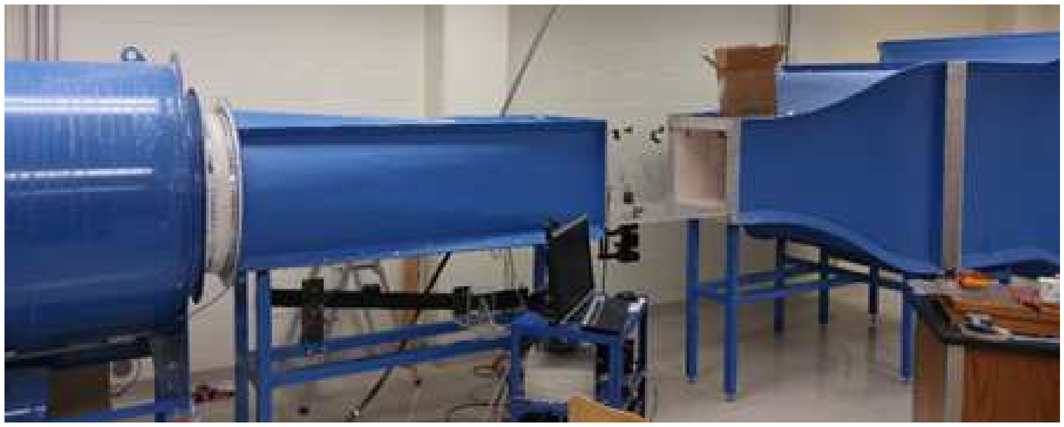

Figure 3.3: Wind Tunnel used for Experimental Work

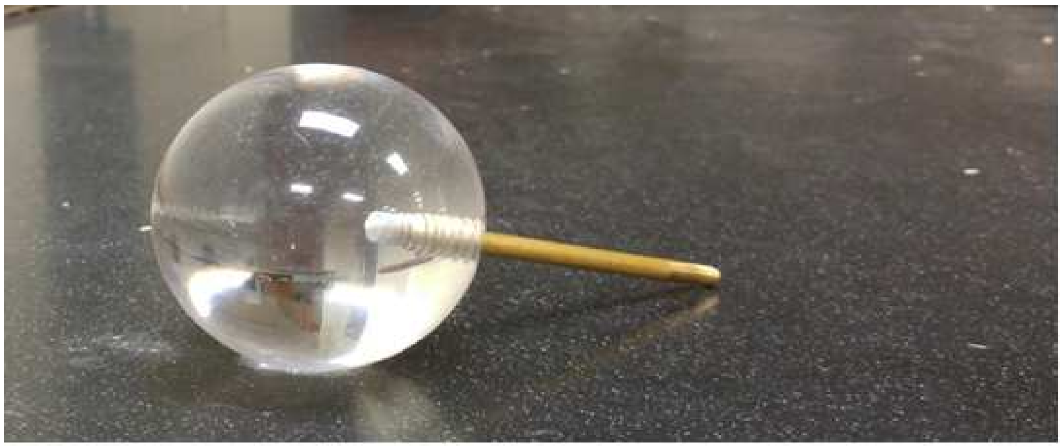

Figure 3.4: Sphere for determining Flow Regime in Wind Tunnel

In order to determine the type of flow coming through the test section, a model sphere with a smooth surface (as shown in Figure 3.4) was placed in the tunnel and the drag was recorded and compared with published data. At the highest speed tested, the Reynolds number was found to be just over 213,000, and the drag coefficient was 0.55, which agrees well with the published data as shown in Figure 3.5 [8]. The Reynolds number is just within the transition region, thus it was expected that both laminar and turbulent flow would be present when testing the vehicle models. In other words, it was expected that flow transition would occur somewhere on the surface of the model placed in the tunnel, which matches the behavior of the flow over the full size vehicle. After the Reynolds number for wind tunnel testing was determined, the CFD simulations were re-run at the same Reynolds number as the wind tunnel to make comparisons, as discussed in section 4.3.

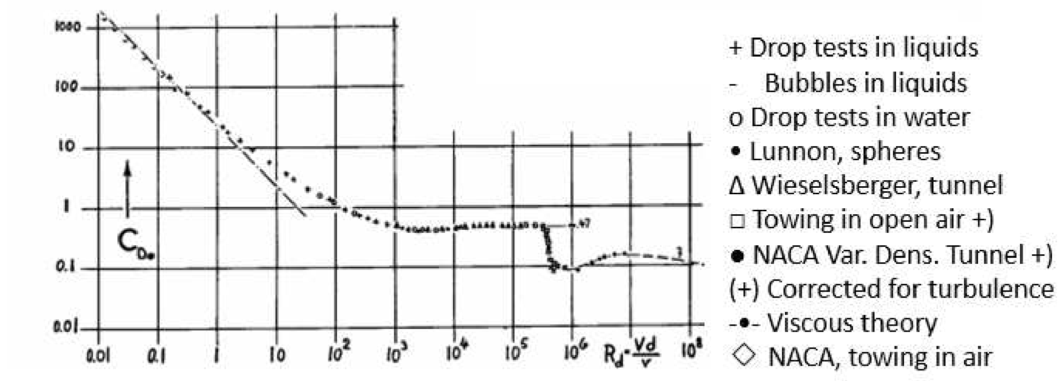

Figure 3.5: Drag on Smooth Sphere from Hoerner [8]

The wind tunnel sensor was connected to a LabView program so that the drag and pitot probe pressure could be monitored and recorded.

The ideal scenario for testing road vehicles in a wind tunnel is to have a moving belt underneath the vehicle that moves at the same speed as the freestream in order to simulate the car driving on the road [2]. However, this capability was not available in the wind tunnel used. Consideration was given to adding a fixed plate underneath the model in order to simulate the ground. After reviewing the literature [8], it was found that the change in drag for a simulation without a fixed plate was the same as that for simulations with a fixed plate in reference to the moving belt scenario. In other words, the fixed plate setup would not move the drag values any closer to the ideal case than the non-fixed plate setup. In the literature cited, the fixed plate setup provided drag values about 3% above the moving belt case for an automobile model, and the non-fixed plate setup provided drag values about 4% below the moving belt case. The fixed plate setup has the added disadvantage of a developing growing boundary layer over the plate. Without any method of removing the boundary layer (such as suction, as described in [2]), the underbody flow of the vehicle would be affected. With these considerations, it was decided that leaving out the fixed plate would be adequate for wind tunnel testing. A discussion of the resulting Reynolds number for the wind tunnel trials and comparison with the CFD results at the same Reynolds number is given in section 4.3.

3.4 Experimental procedure

In order to begin testing the scale models, the sensor in the wind tunnel had to be calibrated. This was done by taking the sensor out of the test section and hanging weights on the sensor in the direction of the drag force. Figure 3.6 shows a photo of sensor calibration as well as a model in the test section. As a result of the calibration, the unit of force that was recorded was grams-force. The sensors consisted of two linear variable displacement transducers (LVDTs) connected to a rod that had a 3/8" hole for screwing stings. A pitot probe was used to determine the freestream velocity according to the equation $V = (2\Delta P/\rho)^{1/2}$, where ΔP is the pressure difference measured by the pitot probe, ρ is the air density, and V is the air speed. ΔP was found by measuring pressure at two locations; the freestream inside the tunnel and outside the tunnel, and then taking the difference. The density was found by looking up the elevation of Provo and using the elevation in the standard atmosphere table [9] to calculate density. Velocity plots from the CFD results were used in order to determine the placement of the pitot probe so that it was in the freestream.

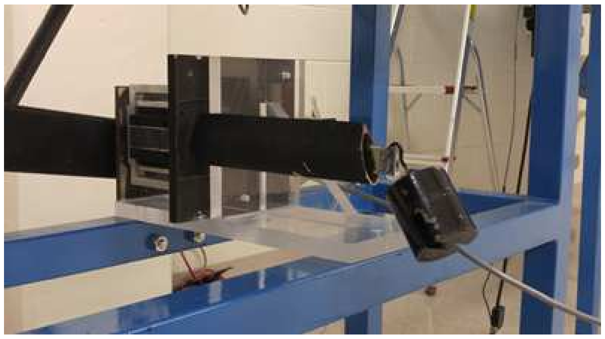
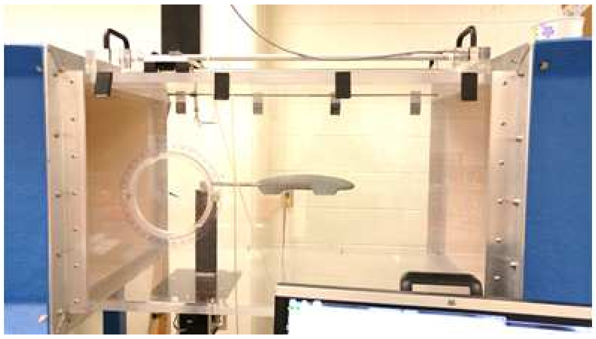

Figure 3.6: Sensor Calibration (left) and Model Attachment (right)

After calibration, the sensor was placed back in the test section and the rod sting was screwed onto the sensor. The sensor was then zeroed from the LabView program. The wind tunnel was turned on and set to run at the desired speed. Once the wind tunnel achieved the desired speed, a minimum of five data points were captured (usually ten or more). These data points included the instantaneous drag in grams-force and the pitot probe pressure in inches of water. The wind tunnel speed was then changed, and new data points were captured. Once data for all speeds was obtained, the process (except calibration) was repeated again with the sting and model attached.

The data was then downloaded and opened in Excel for post-processing. The next model was then attached to the sting, the forces were zeroed, and measurements were taken at the same speeds. Measurements for all three models consisted of one trial run. In order to do another trial run, the sensor was recalibrated and the measurements for each vehicle retaken.

For post-processing, the wind tunnel speed was calculated with the pitot pressure reading. The forces were converted to Newtons, and the drag readings from the ”sting only” configuration were subtracted from the drag readings of the ”sting plus model” configuration. Using the frontal areas from the scaled CAD data, the drag area and coefficient of drag were then calculated. Comparison could then be made between the baseline, Model C, and PacCar II.

3.5 Results

3.5.1 Drag Data

Table 3.1 displays the drag results from testing at the highest speed, along with the 95% confidence interval for the changes in drag from the baseline (see section 3.5.2 for confidence interval calculation). The Reynolds number for the CFD simulations (1.6×10^6) could not be matched exactly in the experiments since the required velocity was beyond the maximum for the wind tunnel. In order to get as close as possible to the CFD Reynolds number, the drag comparisons were done at around 100 mph, giving a Reynolds number around $8.6 \times 10^5 - 8.8 \times 10^5$ depending on the model. The wind tunnel data shows that the drag area for Model C is smaller than the baseline, indicating that the modified shape changes were successful in reducing drag.

Table 3.1: Wind Tunnel Drag Results (Velocity = 100 mph)

Model	**Mean C_D**	**C_D% change from Baseline**	**Mean Drag Area (m^2)**	**Drag area % Change from Baseline**
Baseline	0.099	-	2.60×10^{-4}	-
Model C	0.094	-5.25% $\pm$ 2.68%	2.40×10^{-4}	-7.50 $\pm$ 2.61%
PacCar II	0.096	-2.95% $\pm$ 2.08%	2.15×10^{-4}	-17.29% $\pm$ 1.71%

The mean values for C_D and drag area were generated from the eight trials that were run for all three models. Figure 3.7 shows the drag area data for each trial for the two highest speeds tested in the wind tunnel, along with the frontal area of each scale model. PacCar II has the lowest drag area, a consequence of having the lowest frontal area (16% lower than Model C).

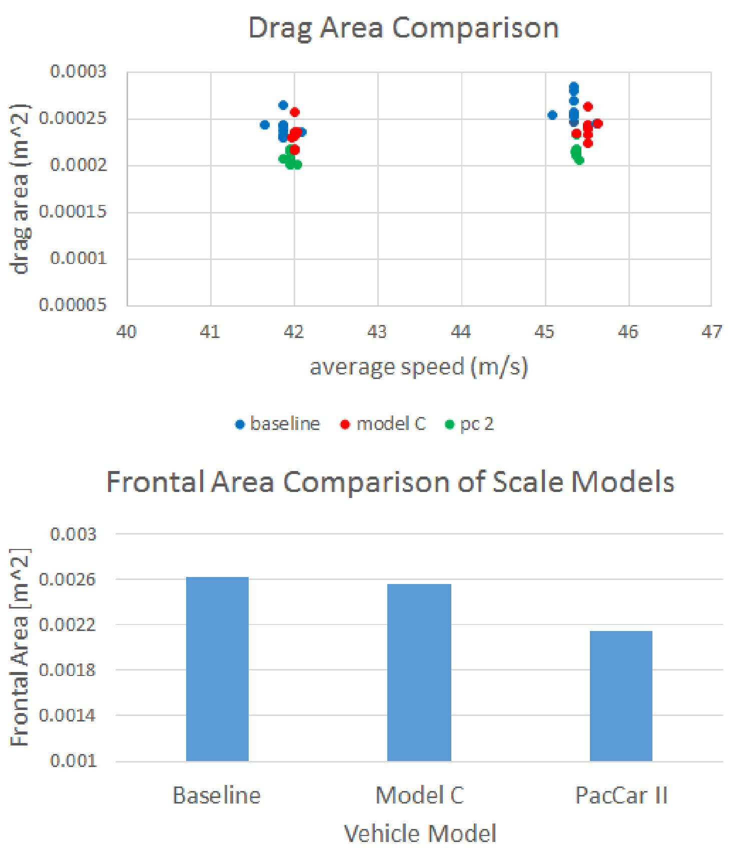

Figure 3.7: Drag Data (top) for Two speeds and Frontal Area Comparison (bottom)

3.5.2 Uncertainty Analysis

Two factors that contributed to the uncertainty in the experiments were the bias error from the sensor and the noise error from repeated measurements at the same velocity. Calibrating the sensor before every trial effectively reduced the bias error so that it could be neglected when compared to the noise in the data. Thus the focus during the uncertainty analysis was quantifying the noise error from repeated measurements. During calibration, the lift sensor was found to be malfunctioning, and therefore lift data was excluded from the measurements. After each calibration, a 500 gram mass was placed in the direction of the drag load in order to assess the bias error, which was observed to be around 0.05 grams force. In order to consider the effects of noise in the data, ΔC_D and $\Delta DragArea$ were calculated from the measurements taken. The changes in drag were then converted to a percentage of the original baseline drag, and the standard deviation of the mean was calculated from the Δ values. The standard deviation of the mean was then used to calculate a 95% confidence interval, which served as the measure of the effect of the wind tunnel noise on the data. As can be seen from Table 3.1, the wind tunnel noise was fairly consistent throughout all the trials. All of the uncertainty calculations were done in Microsoft Excel.

CHAPTER 4. DISCUSSION OF RESULTS

This chapter provides a comparison of the CFD and experimental results. Data from the comparisons are then used to determine the fuel efficiency improvement from the Model C shape. Conclusions are then stated regarding how to increase the fuel efficiency of the vehicle.

4.1 Comparison of Experimental and Numerical Results

Since two length scales were used in the numerical and experimental work, comparisons were divided between full scale results that can be applied to the next iteration of the BYU Supermileage vehicle, and model scale results that originated from wind tunnel testing. Also, two Reynolds numbers were used, which also divided the results into more categories. Table 4.1 below shows the frontal areas of all three vehicles tested, both full scale and model scale. These frontal areas were used when calculating drag for all the results.

Table 4.1: Frontal Areas of Vehicles Used in Simulations and Wind Tunnel Testing

Model	**Frontal Area Full Scale [m^2]**	**Frontal Area Model Scale [m^2]**
Baseline	0.337	0.00262
Model C	0.314	0.00256
PacCar II	0.254	0.00215

4.2 Results for Re = 1.6 Million

A Reynolds Number of 1.6×10^6 was used in the CFD simulations in order to match the Reynolds number in publications for PacCar II [1]. This would help to facilitate comparisons to PacCar II for both C_D and drag area. Figure 4.1 shows the values for C_D and the changes in C_D from the CFD simulations and from the published data values regarding PacCar II [1]. All changes in drag for each figure are measured relative to the baseline vehicle. Figure 4.1 shows that bigger reductions in C_D can be achieved if the flow is laminar, showing the importance of encouraging laminar flow over the surface of the vehicle. PacCar II shows a higher C_D value, although the exact parameters used by PacCar II engineers for their CFD simulations are unknown, so an exact comparison cannot be made. For example, one factor that could increase the drag is the addition of spinning wheels (which was not simulated in this thesis work), and the type of flow specified. The published number for the PacCar II C_D is included in order to serve as a check to make sure that drag results were in the right range.

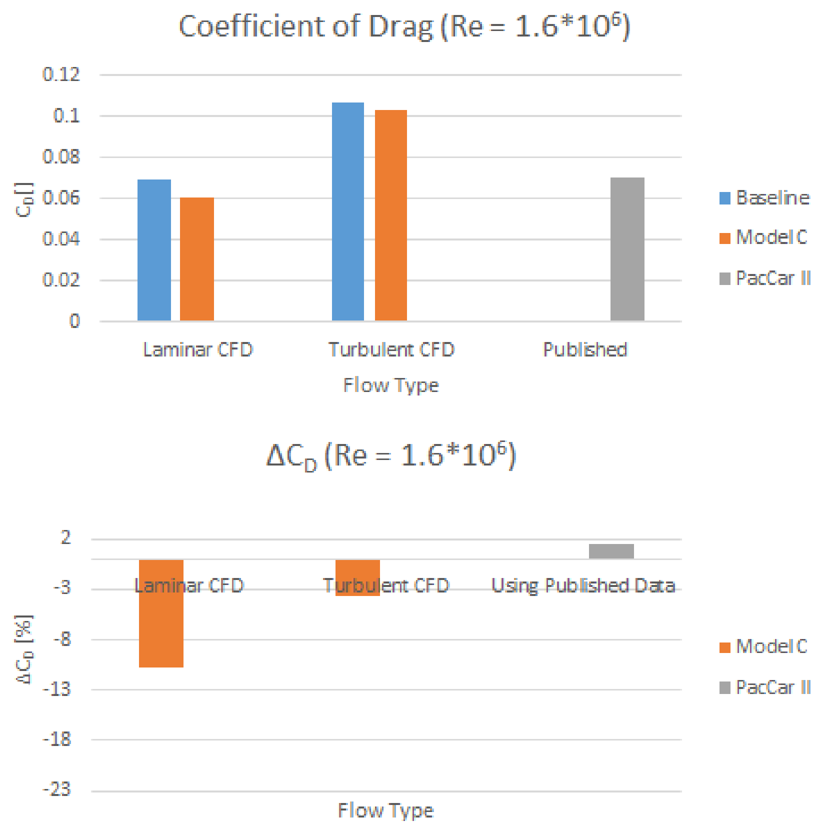

Figure 4.1: C_D Results for Re =1.6×10^6

The same conclusion can be drawn by observing the drag area results at the same Reynolds number. Figure 4.2 shows the drag area for the same Reynolds number using the full scale frontal area. A drag area reduction of 17.4% can be obtained by using the Model C shape if laminar flow is achieved over all of the surface. The PacCar II shape shows the lowest drag area due to the smaller frontal area when compared to Model C. The PacCar II engineers used a fuel cell system to power their vehicle, allowing them to more tightly package their interior components, leading to a smaller frontal area [1]. The reduction in frontal area achieved with Model C is more realistic because Model C and the baseline vehicle use a conventional gasoline engine, which requires more interior space than the fuel cell system used on PacCar II.

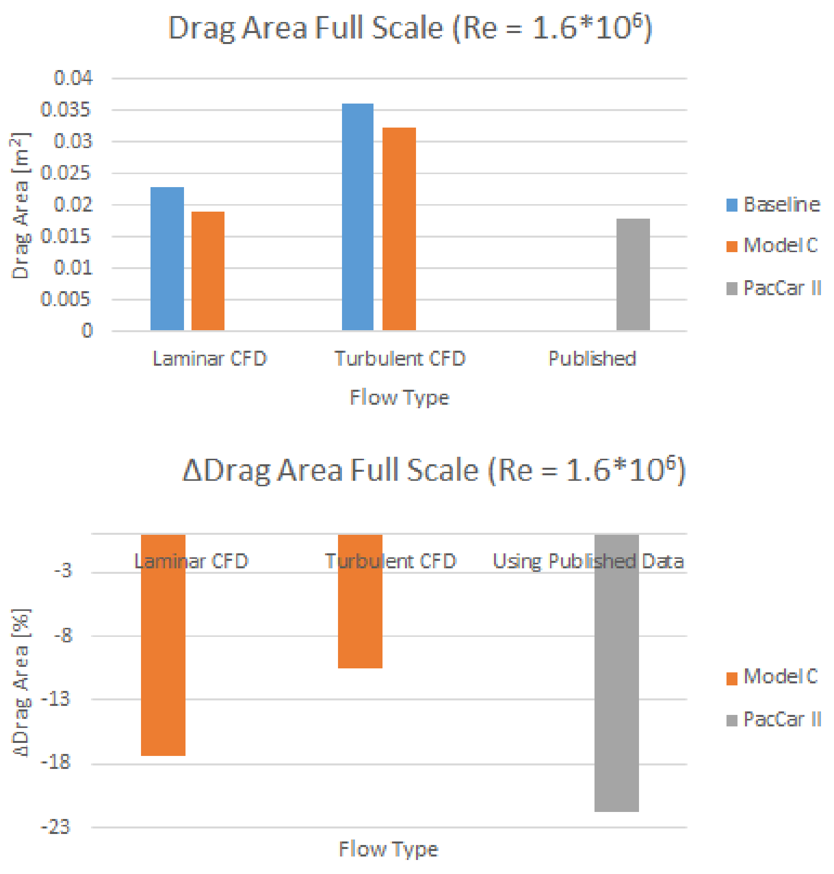

Figure 4.2: Drag Area Results for Re $=1.6 \times 10^6$

4.3 Results for Re = 878,000

The average Reynolds number for the wind tunnel testing for all three models (based on model length) was 878,060. The individual Reynolds numbers for the baseline and Model C were matched in the CFD simulations and compared with the experimental results. Since the CFD simulations provided fully laminar and fully turbulent results (see section 2.2.3), it was expected that the experimental results (with transition actually occurring on the surface) would fall in between the CFD results. Figure 4.3 below shows the C_D results from the experiments and simulations. Again, reductions in C_D of almost 10% can be achieved when using the Model C shape. The reductions are larger when under laminar flow, which again shows the importance of achieving laminar flow in the real case. The experimental values for the baseline and Model C are in between the lamniar and turbulent CFD values, indicating that flow transition occurred somewhere on the surface of the vehicle during the experiments. This transition likely occurred on the rear portion of the vehicle since the curvature of the front portion provides a favorable pressure gradient.

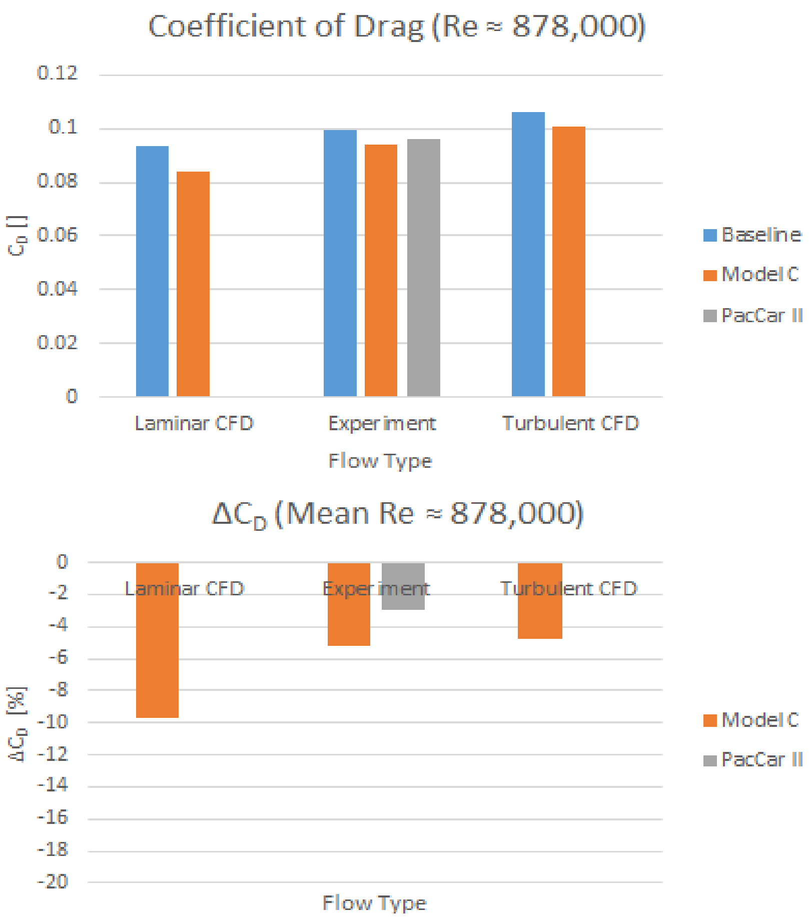

Figure 4.3: C_D Results for Re $=8.78 \times 10^5$

Figures 4.4 and 4.5 show the drag area reductions for both the full scale and model scale vehicles. Again, PacCar II shows the biggest reduction in drag area due to its much smaller frontal area. Bigger drag area reductions are seen with the full scale vehicles. This is most likely due to the fact that the geometry changes between the models are more difficult to differentiate in smaller scale models, and thus the desired changes to drag are harder to achieve.

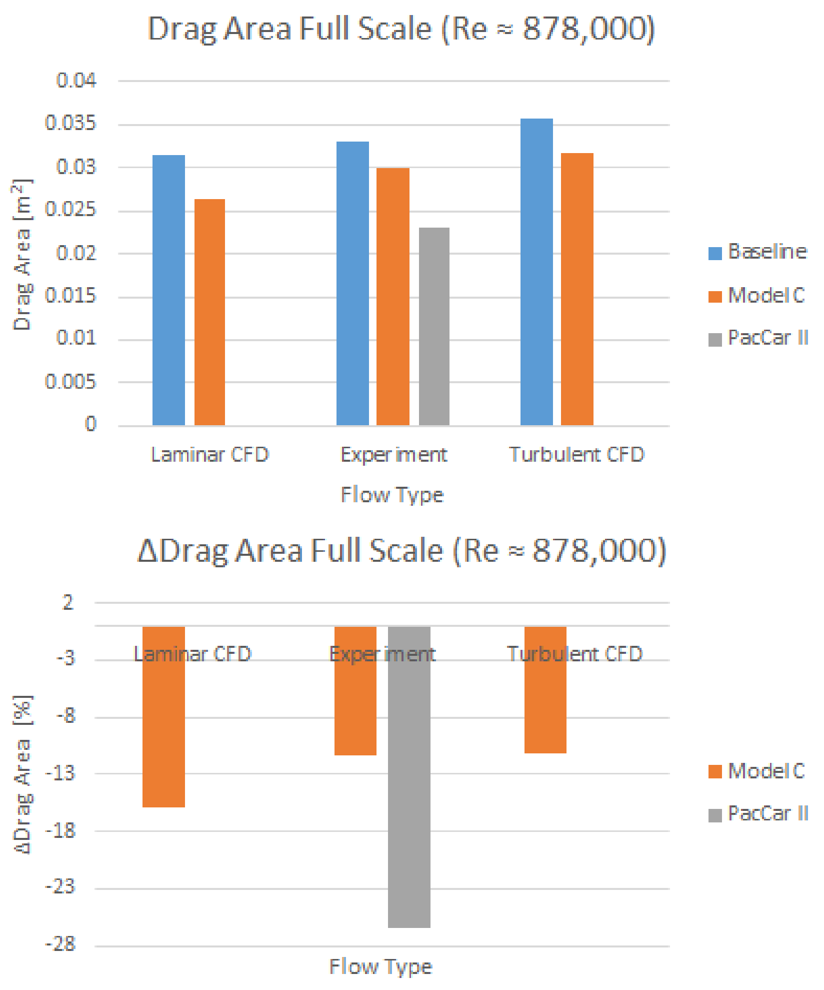

Figure 4.4: Drag Area Results, Full Scale, for Re $=8.78 \times 10^5$

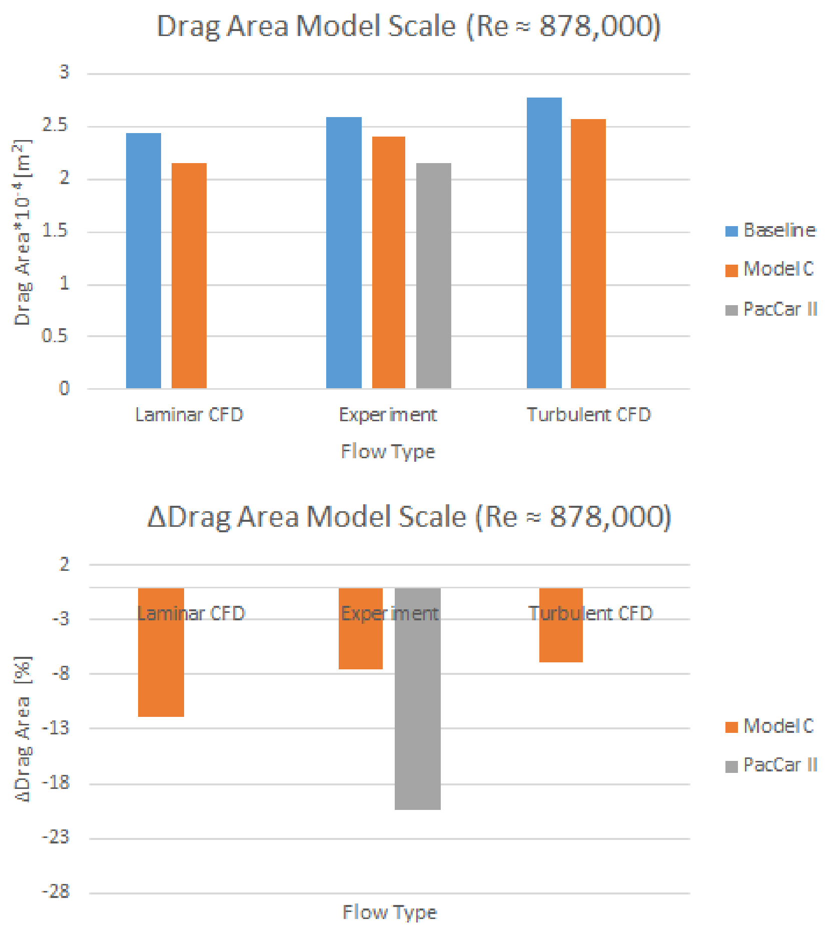

Figure 4.5: Drag Area Results, Model Scale, for Re $=8.78 \times 10^5$

4.4 Impact on Vehicle Fuel Efficiency

The full scale Re drag results shown in Figure 4.2 were used to consider the impact of the aerodynamic improvements on the fuel economy. Table 4.2 shows the estimated MPG ratings assuming fully laminar and fully turbulent flow. The theoretical MPG rating is estimated using a formula based off of the conservation of energy principle, with inputs for engine efficiency, the mass of the vehicle, the rolling resistance coefficient, the vehicle frontal area, and the coefficient of drag. Table 4.2 assumes an engine efficiency of 20%, a rolling resistance coefficient of 0.002, and a vehicle mass of 70 kg. These numbers are targets for next year's vehicle.

Bigger MPG improvements and higher overall MPG ratings can be seen with the laminar results since drag reductions are bigger (478 MPG improvement for laminar versus 296 MPG for

Table 4.2: Estimated MPG Improvement using Full Scale Drag Area Results

Vehicle Model	Fully Laminar Estimated MPG	Fully Turbulent Estimated MPG
Baseline	7385	6221
Model C	7863	6517

turbulent). The real drag area of the BYU Supermileage vehicle is most likely in between the laminar and turbulent value since there is probably transition somewhere on the rear portion. Thus the actual improvement in MPG rating will probably fall in between 296 and 478 MPG, although the MPG rating itself will most likely vary from the theoretical goals. For comparison, the BYU Supermielage team achieved 1244 MPG at the 2015 SAE Supermileage Competition. Table 4.2 indicates that the improvement of the MPG rating will depend on how much of the wetted area is laminar flow. Therefore the main conclusion is that achieving laminar flow over the largest wetted area possible will lead to the biggest improvements in vehicle fuel efficiency. This can be done by changing next year's vehicle shape to match the Model C shape, and also by treating the surface of the vehicle to encourage attached laminar flow. Possible surface treatments include:

- **Surface Polish:** Adding polish to the surface of the vehicle will help to clear any small irregularities on the surface and help get rid of small protuberances that could trip the flow into turbulence. The PacCar II engineers note that applying successive amounts of polish to the surface decreased the drag by 4% [1]. In general, surface roughness serves to decrease the transition Reynolds number required for turbulence. Figure 4.6 below shows a graph of transition Reynolds number as a function of sand roughness for round bodies and also for a flat plate, taken from Hoerner [8]. In the graph, the value *k* represents a typical grain diameter. The graph shows that blunt bodies like the sphere have a weak dependence on surface roughness, but the flat plate shows a stronger dependence. The transition Reynolds number decreases as surface roughness increases, causing an earlier transition to turbulent flow over the body, which is undesired for the Supermileage vehicle. Thus, Figure 4.6 supports the conclusion that, for a streamlined shape like the Supermileage vehicle, efforts should be taken to decrease the surface roughness as much as possible.

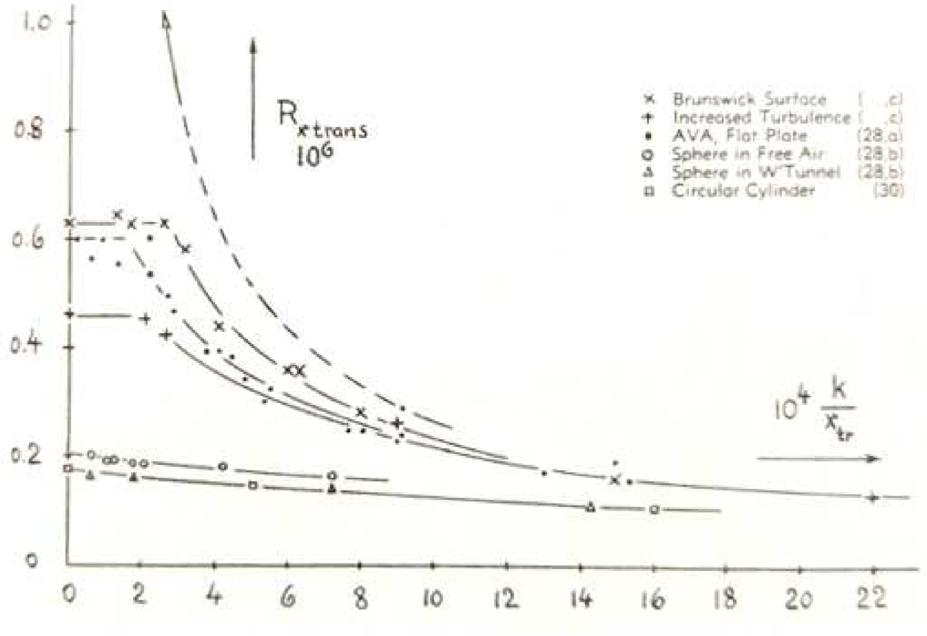

Figure 4.6: Effect of Surface Roughness on Transition Reynolds Number from Hoerner [8]

- **Smooth Gaps and Joints:** Covering gaps on the surface of the vehicle can also help to lower the drag since these possible sources of flow interference are covered. One example of covering joints or gaps is using tape on different parts of the vehicle that are joined together. Figure 4.7 shows an example of tape covering the gap between the body and driver canopy near the front of the vehicle. Another area that could be taped is the gap between the wheels and the cowl. Although the Model C shape is designed with the thinnest possible cowl allowing the wheel to turn, if there are any gaps remaining, they can be taped to lower drag. If multiple

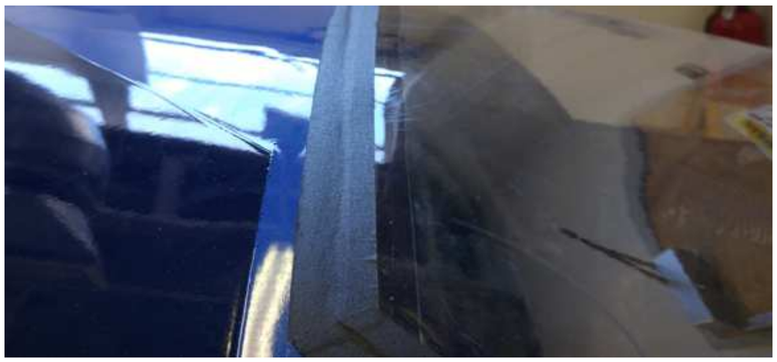

Figure 4.7: Example of Taping Surface to Close Gaps

pieces of tape are used at one location, the second piece of tape should start underneath the first in order to avoid an extra edge of tape exposed to the airflow. PacCar II engineers noted that using tape on the gap between the canopy and the vehicle reduced drag by 4% [1].

Figure 4.8 shows a graphic that displays the main conclusions using the full scale drag area results. The changes in drag area noted in the figure with $\Delta C_D * A$ are in reference to the Model C drag area change from the baseline. The black dotted line in the figure refers to the change in drag area when moving from fully laminar to fully turbulent flow. The green dotted line shows the fuel savings that can be achieved with a change to the Model C shape and fully laminar flow. With the new Model C shape and efforts to encourage laminar flow through surface treatments, the laminar flow fuel efficiency improvement of 478 MPG can be achieved.

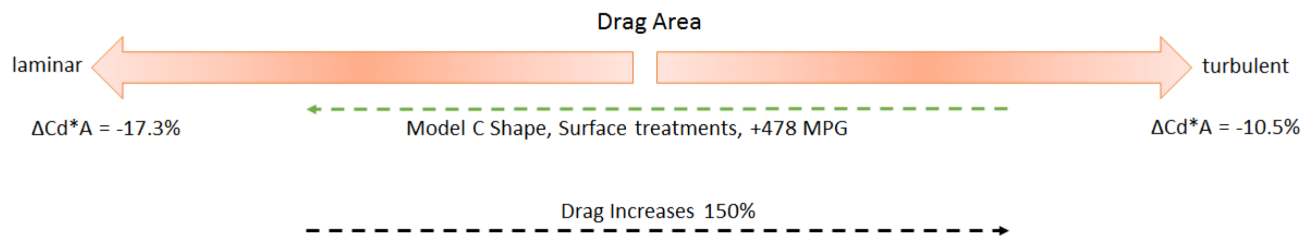

Figure 4.8: Changing the Supermileage Vehicle to Increase Fuel Efficiency

CHAPTER 5. CONCLUSIONS

The objective of this thesis work was to design a new shape for the BYU Supermileage vehicle in order to decrease its aerodynamic drag. Numerical simulations were combined with wind tunnel testing in order to quantify the aerodynamic drag of the current baseline vehicle shape and find a shape with improved aerodynamic performance.

First a CAD model of the baseline vehicle was imported into the CFD program Star CCM+ for simulations. A surface mesh and volume mesh was defined for the vehicle surface. Physics settings were specified and the coefficient of drag and drag area were obtained for the vehicle at a Reynolds number of 1.6×10^6. Then the mesh morphing program Sculptor was used to change the shape of the surface mesh in order to investigate how to decrease the aerodynamic drag. The modified shape from Sculptor was imported into Star CCM+ and further simulations were run. It was found that a 10.8% and 17.4% reduction in the coefficient of drag and drag area, respectively, could be obtained for laminar flow. Similarly, a 3.7% and 10.5% decrease in coefficient of drag and drag area could be achieved with turbulent flow.

Wind tunnel testing was carried out in order to validate the findings from the numerical simulations. Three scale models were printed using a 3D printer, representing the baseline vehicle, Model C, and PacCar II. Testing was carried out in the BYU wind tunnel in the basement of the Clyde building, at a average Reynolds number of 8.78×10^5. The results showed a 5.3% and 7.5% reduction in the coefficient of drag and drag area, respectively, for Model C.

Using the full scale results at Re = 1.6×10^6, it was found that the fuel efficiency of the current vehicle could be improved by 478 MPG with fully laminar flow and 296 MPG with fully turbulent flow. Thus it was concluded that laminar flow should be encouraged over as much of the wetted area as possible in order to reduce the aerodynamic drag. This could be done by utilizing the following recommendations:

- **Change the baseline vehicle shape to match that of Model C:** This will lower the frontal area and the drag area for the vehicle, as shown by the CFD results and the experimental results.
- **Apply surface polish and tape gaps:** This will delay the transition to turbulent flow so that laminar flow can be achieved over a larger wetted area of the vehicle.

If there is a desire to continue this research, there are a few areas that could be investigated further:

- **Adding spinning wheels to the CFD simulations:** Adding spinning wheels introduces another source of drag to the vehicle [2]. Investigations could be done on different size wheels at different angles to the road to see its effect on the overall drag to the vehicle.
- **Designing a new shape using airfoil sections:** If there is a need to design a new vehicle shape from scratch, airfoil sections could be used as a basis for the basic body of the vehicle. for example, NACA airfoils could be used for the main body, while laminar type airfoils could be used for the wheel cowls. Any new design must take into consideration the manufacturing capabilities of the capstone team.
- **Larger models for experimental testing:** Larger models could be used for the experimental portion and flow visualization could be done in order to find the approximate location of flow separation at a desired yaw angle. The PacCar II engineers used a 1:2 scale model for their wind tunnel tests and were able to match the Reynolds number of the full scale vehicle. The larger model size also allowed them to use flow visualization techniques. [1]

REFERENCES

[1] Santin, J., Onder, C., Bernard, J., Isler, D., Kobler, P., Kolb, F., Weidmann, N., and Guzzella, L., 2007. *The World's Most Fuel Efficient Vehicle: Design and Development of PacCar II.*, 1 ed., Vol. 1 vdf Hochschulverlag AG an der ETH Zurich, Zurich. 1, 4, 18, 24, 45, 47, 51, 53, 56

[2] Hucho, W., 1998. *Aerodynamics of Road Vehicles.*, 2 ed. SAE International, Warrendale, PA. 3, 38, 40, 56

[3] Maji, S., and Almadi, H., 2007. "Optimization of the aerodynamic design of the supermileage vehicle." *SAE Technical Paper 2007-01-0901.* 3, 4

[4] Gagnon, L., Richard, M., Beardsell, G., and Boudreau, M., 2012. "The process of making and aerodynamically efficient car body for the sae supermileage competition." *SAE International 2012-01-0176.* 4, 5

[5] CDAdapco, 2015. Best practices for vehicle external aerodynamics. Provided by Chris Penny at CD Adapco (chris.penny@cd-adapco.com). 8, 9, 11, 15

[6] CDAdapco, 2014. User guide star ccm+ version 9.02. Available with copy of Star CCM+ V. 9.02. 13

[7] Fox, R. W., McDonald, A. T., and Pritchard, P. J., 2004. *Introduction to Fluid Mechanics.*, 6 ed. John Wiley and Sons, Inc., USA. 21, 34, 35

[8] Hoerner, S., 1965. *Fluid-Dynamic Drag.*, 1 ed. SF Hoerner, New Jersey. 21, 39, 40, 51, 52

[9] Anderson, J. D., 2011. *Fundamentals of Aerodynamics.*, 5 ed. McGraw Hill, USA. 21, 41

[10] NASA, 2008. Uncertainty and error in cfd simulations. from NPARC Alliance CFD Verificaiton and Validation Website., 2008 http://www.grc.nasa.gov. 22

[11] NASA, 2008. Examining spatial (grid) convergence. from NPARC Alliance CFD Verificaiton and Validation Website., 2008 http://www.grc.nasa.gov. 22, 59

[12] Roache, P., 1997. "Quantification of uncertainity in computational fluid dynamics." *Annual Review of Fluid Mechanics,* **29**, pp. 123–60. 22

APPENDIX A. BATCH SCRIPT AND JAVA MACRO FOR RUNNING SIMULATIONS ON THE BYU SUPERCOMPUTER

A.1 Batch Script Example

The code below is for submitting simulations to be run on the BYU Supercomputer. The requested resources such as memory and number of nodes are specified and the appropriate simulation and macro files are also listed so the Supercomputer can access the right files for simulations. The process starts with an empty Star CCM+ simulation file named ”Empty_forimporting”. Once the job is started and a Star CCM+ license is activated, the associated java macro is called in order to apply all the simulation settings. In this case the java macro is called ”yplusfix_importsurf_run _June20.java”. The job is run until the simulation completes, the time limit is reached, or an error occurs. An email is then sent to the user notifying that the job is complete. This script also allows for the submitting of multiple jobs at once, by submitting an array of jobs. The job scheduler assigns one job number to the array of jobs, followed by an underscore. Thus an array of jobs would have numbers such as 23_1, 23_2, etc.

Batch Script

```
#!/bin/bash

#Submit this script with: sbatch thefilename

#SBATCH -J "basethick"      # Job Name
#SBATCH --time=10:30:00     # walltime
#SBATCH --ntasks=32    # number of processor cores (i.e. tasks)
#SBATCH --mem-per-cpu=8G    # memory per CPU
#SBATCH --mail-user=bbobbj@gmail.com    # email address
```

```
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=END
#SBATCH --mail-type=FAIL
#SBATCH --licenses=starccm_ccmpsuite:1
##SBATCH --gid=fslg_pulseturbine            # make it so group
   members can view/edit output files
##SBATCH --nodes=32   # number of nodes
## -C 'ib'   # features syntax (use quotes): -C 'a&b&c&d'

SIM_FILE=Empty_forimporting # simulation name
SIM_DIR=~/compute/simulation/baseline_thicknesschange/runs${
   SLURM_ARRAY_TASK_ID}

# Set the output permissions on files written from my script to
   be 0660 and directories to be 0770
umask 0007



cd ${SIM_DIR} # change the working directory
printf "Job debugging info\n____\n"
printf "Node List"

module load starccm+/9.02.007

echo "Machinefile:"
machinefile=$(/fslapps/fslutils/generate_mpich_machinefile)
cat "$machinefile"
echo "----"

starccm+ \
    -mpi intel \
    -mpiflags "-bootstrap slurm" \
    -np $SLURM_NTASKS \
    -machinefile "$machinefile" \
        -batch yplusfix_importsurf_run_June20.java $SIM_FILE

printf "____\n"
printf "Ending starccm+ run at "
date
printf "\n____\n"
printf "Removing temp files\n"
rm -vf ${SIM_DIR}/*~
printf "____\n"

rm "$machinefile"
exit 0
```

A.2 Java Macro Example

The code below is an example of the commands created by Star CCM+ as part of the automated macro function. Once a macro begins recording in Star CCM+, all the parameters that the user sets as part of setting up a simulation are recorded in the Java programming language. Once all the parameters are set, the macro can then be saved and called in batch mode when running on the Supercomputer. In the code below, a simulation is setup to run on the baseline vehicle. A nastran surface mesh of the baseline vehicle is first imported, and then all of the simulation parameters are defined, including mesh and physics settings. The commands for running and post process are then called. The java macro is included as part of the batch script that is submitted to the Supercomputer.

Java Macro

```
// STAR-CCM+ macro: yplusfix_importsurf_run_June20.java
// Written by STAR-CCM+ 9.02.007
package macro;

import java.util.*;

import star.material.*;
import star.common.*;
import star.base.neo.*;
import star.vis.*;
import star.base.report.*;
import star.flow.*;
import star.trimmer.*;
import star.prismmesher.*;
import star.segregatedflow.*;
import star.metrics.*;
import star.meshing.*;

public class yplusfix_importsurf_run_June20 extends StarMacro {

  public void execute() {
    execute0();
    execute1();
  }

  private void execute0() {

    Simulation simulation_0 =
      getActiveSimulation();

    Units units_0 =
  simulation_0.getUnitsManager().getPreferredUnits(new
     IntVector(new int[]
  {0, 1, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0}));

ImportManager importManager_0 =
  simulation_0.getImportManager();

importManager_0.importNastranSurface(resolvePath("6
   Msurf_Fixed_April16.nas"),
"OneBoundaryPerPatch", true, units_0);

SurfaceRep surfaceRep_0 =
  ((SurfaceRep) simulation_0.getRepresentationManager().
     getObject("Import"));

Region region_0 =
  simulation_0.getRegionManager().getRegion("Region 1");

surfaceRep_0.generateMeshReport(new NeoObjectVector(new
   Object[] {region_0}));

simulation_0.getSceneManager().createGeometryScene("Geometry
Scene", "Outline", "Geometry", 1);

Scene scene_0 =
  simulation_0.getSceneManager().getScene("Geometry Scene 1")
     ;

scene_0.initializeAndWait();
```

```
PartDisplayer partDisplayer_1 =
  ((PartDisplayer) scene_0.getCreatorDisplayer());

partDisplayer_1.initialize();

PartDisplayer partDisplayer_0 =
  ((PartDisplayer) scene_0.getDisplayerManager().getDisplayer
     ("Outline 1"));

partDisplayer_0.initialize();

PartDisplayer partDisplayer_2 =
  ((PartDisplayer) scene_0.getDisplayerManager().getDisplayer
     ("Geometry 1"));

partDisplayer_2.initialize();

scene_0.open(true);

CurrentView currentView_0 =
  scene_0.getCurrentView();

currentView_0.setInput(new DoubleVector(new double[]
{-5.2009992599487305, -7.169999927282333,
   -7.345000013709068}),
new DoubleVector(new double[] {-5.2009992599487305,
-7.169999927282333, 91.89512116659364}), new DoubleVector(new
double[] {0.0, 1.0, 0.0}), 25.906870266242805, 0);
```

```
scene_0.setTransparencyOverrideMode(1);

scene_0.setTransparencyOverrideMode(1);

MeshPartFactory meshPartFactory_0 =
  simulation_0.get(MeshPartFactory.class);

SimpleBlockPart simpleBlockPart_0 =

  meshPartFactory_0.createNewBlockPart
  (simulation_0.get(SimulationPartManager.class));

simpleBlockPart_0.setDoNotRetessellate(true);

Coordinate coordinate_0 =
  simpleBlockPart_0.getCorner1();

LabCoordinateSystem labCoordinateSystem_0 =
  simulation_0.getCoordinateSystemManager().
     getLabCoordinateSystem();

coordinate_0.setCoordinateSystem(labCoordinateSystem_0);

Coordinate coordinate_1 =
  simpleBlockPart_0.getCorner2();

coordinate_1.setCoordinateSystem(labCoordinateSystem_0);

coordinate_0.setCoordinate(units_0, units_0, units_0, new
DoubleVector(new double[] {-1.0, -1.0, -1.0}));
```

```
coordinate_1.setCoordinate(units_0, units_0, units_0, new
DoubleVector(new double[] {1.0, 1.0, 1.0}));

coordinate_0.setValue(new DoubleVector(new double[] {-12.281,
-2.8865, -1.977}));

coordinate_1.setValue(new DoubleVector(new double[] {7.95,
0.32, 0.30808}));

simpleBlockPart_0.setCoordinateSystem(labCoordinateSystem_0);

simpleBlockPart_0.rebuildSimpleShapePart();

simpleBlockPart_0.setDoNotRetessellate(false);

scene_0.setTransparencyOverrideMode(0);

simpleBlockPart_0.setPresentationName("near field");

scene_0.setTransparencyOverrideMode(1);

scene_0.setTransparencyOverrideMode(1);

SimpleBlockPart simpleBlockPart_1 =

  meshPartFactory_0.createNewBlockPart
  (simulation_0.get(SimulationPartManager.class));

simpleBlockPart_1.setDoNotRetessellate(true);
```

```
Coordinate coordinate_2 =
  simpleBlockPart_1.getCorner1();

coordinate_2.setCoordinateSystem(labCoordinateSystem_0);

Coordinate coordinate_3 =
  simpleBlockPart_1.getCorner2();

coordinate_3.setCoordinateSystem(labCoordinateSystem_0);

coordinate_2.setCoordinate(units_0, units_0, units_0, new
DoubleVector(new double[] {-1.0, -1.0, -1.0}));

coordinate_3.setCoordinate(units_0, units_0, units_0, new
DoubleVector(new double[] {1.0, 1.0, 1.0}));

coordinate_2.setValue(new DoubleVector(new double[] {-16.797,
-6.5, -4.015}));

coordinate_3.setValue(new DoubleVector(new double[] {11.1595,
0.32, 0.027}));

simpleBlockPart_1.setCoordinateSystem(labCoordinateSystem_0);

simpleBlockPart_1.rebuildSimpleShapePart();

simpleBlockPart_1.setDoNotRetessellate(false);

scene_0.setTransparencyOverrideMode(0);
```

```
simpleBlockPart_1.setPresentationName("mid field");

MeshContinuum meshContinuum_0 =
  ((MeshContinuum)
  simulation_0.getContinuumManager().getContinuum("Mesh 1"));

meshContinuum_0.enable(TrimmerMeshingModel.class);

meshContinuum_0.enable(PrismMesherModel.class);

PrismMesherModel prismMesherModel_0 =

  meshContinuum_0.getModelManager().getModel(PrismMesherModel
     .class);

prismMesherModel_0.setBoundaryMarchAngle(85.0);

prismMesherModel_0.setConcaveAngleLimit(1.0);

prismMesherModel_0.setConcaveAngleLimit(0.0);

prismMesherModel_0.setConvexAngleLimit(359.0);

prismMesherModel_0.setConvexAngleLimit(360.0);

prismMesherModel_0.setNearCoreLayerAspectRatio(1.0);

Units units_1 =
  ((Units) simulation_0.getUnitsManager().getObject("mm"));

meshContinuum_0.getReferenceValues().get(BaseSize.class).
    setUnits(units_1);

meshContinuum_0.getReferenceValues().get(BaseSize.class).
    setValue(14.0);

MaximumCellSize maximumCellSize_0 =
  meshContinuum_0.getReferenceValues().get(MaximumCellSize.
      class);

GenericRelativeSize genericRelativeSize_0 =

  ((GenericRelativeSize) maximumCellSize_0.getRelativeSize())
      ;

genericRelativeSize_0.setPercentage(3600.0);

NumPrismLayers numPrismLayers_0 =

  meshContinuum_0.getReferenceValues().get(NumPrismLayers.
      class);

numPrismLayers_0.setNumLayers(12);

PrismLayerStretching prismLayerStretching_0 =

  meshContinuum_0.getReferenceValues().get(
      PrismLayerStretching.class);
```

```
prismLayerStretching_0.setStretching(1.3);

PrismThickness prismThickness_0 =

  meshContinuum_0.getReferenceValues().get(PrismThickness.
     class);
prismThickness_0.getRelativeOrAbsoluteOption()
.setSelected(RelativeOrAbsoluteOption.ABSOLUTE);

GenericAbsoluteSize genericAbsoluteSize_0 =

  ((GenericAbsoluteSize) prismThickness_0.getAbsoluteSize());

genericAbsoluteSize_0.getValue().setUnits(units_1);

genericAbsoluteSize_0.getValue().setValue(24.0);

SurfaceSize surfaceSize_0 =
  meshContinuum_0.getReferenceValues().get(SurfaceSize.class)
     ;

RelativeTargetSize relativeTargetSize_0 =
  surfaceSize_0.getRelativeTargetSize();

relativeTargetSize_0.setPercentage(3000.0);

VolumeSource volumeSource_0 =
  meshContinuum_0.getVolumeSources().createVolumeSource();
```

```
volumeSource_0.getPartGroup().setObjects(simpleBlockPart_1);

TrimmerSizeOption trimmerSizeOption_0 =
  volumeSource_0.get(MeshConditionManager.class).get(
     TrimmerSizeOption.class);

trimmerSizeOption_0.setTrimmerSizeOption(true);

VolumeSourceSize volumeSourceSize_0 =
  volumeSource_0.get(MeshValueManager.class).get(
     VolumeSourceSize.class);

GenericRelativeSize genericRelativeSize_1 =
  ((GenericRelativeSize) volumeSourceSize_0.getRelativeSize()
     );

genericRelativeSize_1.setPercentage(1000.0);

volumeSource_0.setPresentationName("mid field");

VolumeSource volumeSource_1 =
  meshContinuum_0.getVolumeSources().createVolumeSource();

volumeSource_1.getPartGroup().setObjects(simpleBlockPart_0);

TrimmerSizeOption trimmerSizeOption_1 =
  volumeSource_1.get(MeshConditionManager.class).get(
     TrimmerSizeOption.class);

trimmerSizeOption_1.setTrimmerSizeOption(true);
```

```
VolumeSourceSize volumeSourceSize_1 =
  volumeSource_1.get(MeshValueManager.class).get(
     VolumeSourceSize.class);

GenericRelativeSize genericRelativeSize_2 =
  ((GenericRelativeSize) volumeSourceSize_1.getRelativeSize()
     );

genericRelativeSize_2.setPercentage(500.0);

volumeSource_1.setPresentationName("near field");

PhysicsContinuum physicsContinuum_0 =
  simulation_0.getContinuumManager().createContinuum(
     PhysicsContinuum.class);

physicsContinuum_0.enable(ThreeDimensionalModel.class);

physicsContinuum_0.enable(SteadyModel.class);

physicsContinuum_0.enable(SingleComponentGasModel.class);

physicsContinuum_0.enable(SegregatedFlowModel.class);

physicsContinuum_0.enable(ConstantDensityModel.class);

physicsContinuum_0.enable(LaminarModel.class);

VelocityProfile velocityProfile_0 =
  physicsContinuum_0.getInitialConditions().get(
     VelocityProfile.class);

velocityProfile_0.getMethod(ConstantVectorProfileMethod.class
   )
.getQuantity().setComponents(-8.61,0.0, 0.0);

WakeRefinementSet wakeRefinementSet_0 =
  region_0.get(MeshValueManager.class).get(
     WakeRefinementSetManager.class)
  .createWakeRefinementSet();

Boundary boundary_0 =
  region_0.getBoundaryManager().getBoundary("Subtract.Body");

wakeRefinementSet_0.getBoundaryFeatureCurveGroup().setObjects
   (boundary_0);

wakeRefinementSet_0.getDistance().setValue(5.0);

wakeRefinementSet_0.getDirection().setComponents(-1.0, 0.0,
   0.0);

RelativeWakeRefinementSize relativeWakeRefinementSize_0 =
  wakeRefinementSet_0.getRelativeRefSize();

relativeWakeRefinementSize_0.setPercentage(50.0);

Boundary boundary_1 =
  region_0.getBoundaryManager().getBoundary("Subtract.Floor")
     ;

boundary_1.getConditions().get(WallSlidingOption.class)
.setSelected(WallSlidingOption.VECTOR);

WallVelocityProfile wallVelocityProfile_0 =
  boundary_1.getValues().get(WallVelocityProfile.class);

wallVelocityProfile_0.getMethod(ConstantVectorProfileMethod
.class).getQuantity().setComponents(-8.61,0.0, 0.0);

Boundary boundary_2 =
  region_0.getBoundaryManager().getBoundary("Subtract.Inlet")
     ;

VelocityMagnitudeProfile velocityMagnitudeProfile_0 =
boundary_2.getValues().get(VelocityMagnitudeProfile.class);

velocityMagnitudeProfile_0.getMethod
(ConstantScalarProfileMethod.class).getQuantity().setValue
   (8.61);

SegregatedFlowSolver segregatedFlowSolver_0 =
  ((SegregatedFlowSolver)
  simulation_0.getSolverManager().getSolver(
     SegregatedFlowSolver.class));

VelocitySolver velocitySolver_0 =
  segregatedFlowSolver_0.getVelocitySolver();
```

```
velocitySolver_0.setUrf(0.2);

PressureSolver pressureSolver_0 =
  segregatedFlowSolver_0.getPressureSolver();

pressureSolver_0.setUrf(0.2);

pressureSolver_0.setUrf(0.3);

StepStoppingCriterion stepStoppingCriterion_0 =
  ((StepStoppingCriterion)simulation_0.
     getSolverStoppingCriterionManager()
  .getSolverStoppingCriterion("Maximum Steps"));

stepStoppingCriterion_0.setMaximumNumberSteps(4000);



MeshPipelineController meshPipelineController_0 =
  simulation_0.get(MeshPipelineController.class);

meshPipelineController_0.generateVolumeMesh();

FrontalAreaReport frontalAreaReport_0 =
  simulation_0.getReportManager().createReport(
     FrontalAreaReport.class);

Coordinate coordinate_4 =
  frontalAreaReport_0.getViewUpCoordinate();
```

```
coordinate_4.setCoordinate(units_0, units_0, units_0, new
DoubleVector(new double[] {0.0, 0.0, 1.0}));

Coordinate coordinate_5 =
  frontalAreaReport_0.getNormalCoordinate();

coordinate_5.setCoordinate(units_0, units_0, units_0, new
DoubleVector(new double[] {-1.0, 0.0, 0.0}));

frontalAreaReport_0.getParts().setObjects(boundary_0);

ForceCoefficientReport forceCoefficientReport_0 =
  simulation_0.getReportManager().createReport(
     ForceCoefficientReport.class);

forceCoefficientReport_0.getDirection().setComponents(-1.0,
   0.0, 0.0);

forceCoefficientReport_0.getReferenceDensity().setValue
   (1.18415);

forceCoefficientReport_0.getReferenceVelocity().setValue
   (8.61);

Units units_2 =
  simulation_0.getUnitsManager().getInternalUnits(new
  IntVector(new int[] {0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}));
```

```
forceCoefficientReport_0.getReferenceArea().setDefinition("
    $FrontalArea1Report");

forceCoefficientReport_0.getParts().setObjects(boundary_0);

forceCoefficientReport_0.setPresentationName("Cd");

ForceCoefficientReport forceCoefficientReport_1 =
   simulation_0.getReportManager().createReport(
       ForceCoefficientReport.class);

forceCoefficientReport_1.getDirection().setComponents(0.0,
    0.0, -1.0);

forceCoefficientReport_1.getReferenceDensity().setValue
    (1.18415);

forceCoefficientReport_1.getReferenceVelocity().setValue
    (8.61);

forceCoefficientReport_1.getReferenceArea().setDefinition("
    $FrontalArea1Report");

forceCoefficientReport_1.getParts().setObjects(boundary_0);

forceCoefficientReport_1.setPresentationName("Cl");

simulation_0.getMonitorManager().createMonitorAndPlot(new
NeoObjectVector(new Object[] {forceCoefficientReport_0}),
    true,
"%1$s Plot");

ReportMonitor reportMonitor_0 =
  ((ReportMonitor)
  simulation_0.getMonitorManager().getMonitor("Cd Monitor"));

MonitorPlot monitorPlot_0 =
  simulation_0.getPlotManager().createMonitorPlot(new
  NeoObjectVector(new Object[] {reportMonitor_0}), "Cd
     Monitor
  Plot");

simulation_0.getMonitorManager().createMonitorAndPlot(new
NeoObjectVector(new Object[] {forceCoefficientReport_1}),
   true,
"%1$s Plot");

ReportMonitor reportMonitor_1 =
  ((ReportMonitor)
  simulation_0.getMonitorManager().getMonitor("Cl Monitor"));

MonitorPlot monitorPlot_1 =
  simulation_0.getPlotManager().createMonitorPlot(new
  NeoObjectVector(new Object[] {reportMonitor_1}), "Cl
     Monitor
  Plot");

FieldMeanMonitor fieldMeanMonitor_0 =
  simulation_0.getMonitorManager().createMonitor(
     FieldMeanMonitor.class);
```

```
fieldMeanMonitor_0.getParts().setObjects(boundary_0);

StarUpdate starUpdate_0 =
  fieldMeanMonitor_0.getStarUpdate();

starUpdate_0.getUpdateModeOption().setSelected(
   StarUpdateModeOption.ITERATION);

fieldMeanMonitor_0.setSlidingWindow(true);

PrimitiveFieldFunction primitiveFieldFunction_0 =
  ((PrimitiveFieldFunction)
  simulation_0.getFieldFunctionManager().getFunction("
     CdReport"));

fieldMeanMonitor_0.setFieldFunction(primitiveFieldFunction_0)
   ;

fieldMeanMonitor_0.getSlidingWindowOption().setValue(600);

fieldMeanMonitor_0.setPresentationName("mean Cd");

MaxReport maxReport_0 =
  simulation_0.getReportManager().createReport(MaxReport.
     class);

PrimitiveFieldFunction primitiveFieldFunction_1 =
  ((PrimitiveFieldFunction)
  simulation_0.getFieldFunctionManager().getFunction("
     meanCdMonitor"));

maxReport_0.setScalar(primitiveFieldFunction_1);

maxReport_0.getParts().setObjects(boundary_0);

maxReport_0.setPresentationName("average Cd");

FieldMeanMonitor fieldMeanMonitor_1 =
  simulation_0.getMonitorManager().createMonitor(
     FieldMeanMonitor.class);

fieldMeanMonitor_1.getParts().setObjects(boundary_0);

StarUpdate starUpdate_1 =
  fieldMeanMonitor_1.getStarUpdate();

starUpdate_1.getUpdateModeOption().setSelected(
   StarUpdateModeOption.ITERATION);

fieldMeanMonitor_1.setSlidingWindow(true);

PrimitiveFieldFunction primitiveFieldFunction_2 =
  ((PrimitiveFieldFunction)
  simulation_0.getFieldFunctionManager().getFunction("
     ClReport"));

fieldMeanMonitor_1.setFieldFunction(primitiveFieldFunction_2)
    ;
```

```
fieldMeanMonitor_1.getSlidingWindowOption().setValue(600);

fieldMeanMonitor_1.setPresentationName("mean Cl");

MaxReport maxReport_1 =
  simulation_0.getReportManager().createReport(MaxReport.
     class);

PrimitiveFieldFunction primitiveFieldFunction_3 =
  ((PrimitiveFieldFunction)
  simulation_0.getFieldFunctionManager().getFunction("
     meanClMonitor"));

maxReport_1.setScalar(primitiveFieldFunction_3);

maxReport_1.getParts().setObjects(boundary_0);

maxReport_1.setPresentationName("average Cl");

simulation_0.getSceneManager().createScalarScene("Scalar
Scene", "Outline", "Scalar");

Scene scene_1 =
  simulation_0.getSceneManager().getScene("Scalar Scene 1");

scene_1.initializeAndWait();

FixedAspectAnnotationProp fixedAspectAnnotationProp_0 =
  ((FixedAspectAnnotationProp)
  scene_0.getAnnotationPropManager().getAnnotationProp("Logo
     "));

fixedAspectAnnotationProp_0.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_0.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

PartDisplayer partDisplayer_4 =
  ((PartDisplayer) scene_1.getCreatorDisplayer());

partDisplayer_4.initialize();

PartDisplayer partDisplayer_3 =
  ((PartDisplayer) scene_1.getDisplayerManager().getDisplayer
     ("Outline 1"));

partDisplayer_3.initialize();

ScalarDisplayer scalarDisplayer_0 =
  ((ScalarDisplayer) scene_1.getDisplayerManager().
     getDisplayer("Scalar 1"));

scalarDisplayer_0.initialize();

scene_1.open(true);

CurrentView currentView_1 =
  scene_1.getCurrentView();
```

```
currentView_1.setInput(new DoubleVector(new double[]
{-5.2009992599487305, -7.169999927282333,
   -7.345000013709068}),
new DoubleVector(new double[] {-5.2009992599487305,
-7.169999927282333, 91.89512116659364}), new DoubleVector(new
double[] {0.0, 1.0, 0.0}), 25.906870266242805, 0);

Boundary boundary_3 =
  region_0.getBoundaryManager().getBoundary("Subtract.
     SymPlane");

scalarDisplayer_0.getParts().setObjects(boundary_3);

PrimitiveFieldFunction primitiveFieldFunction_4 =
  ((PrimitiveFieldFunction)
  simulation_0.getFieldFunctionManager().getFunction("
     Velocity"));

VectorMagnitudeFieldFunction vectorMagnitudeFieldFunction_0 =
  ((VectorMagnitudeFieldFunction)
  primitiveFieldFunction_4.getMagnitudeFunction());

scalarDisplayer_0.getScalarDisplayQuantity()
.setFieldFunction(vectorMagnitudeFieldFunction_0);

scalarDisplayer_0.setFillMode(1);

scene_1.setPresentationName("scalar velocity");
```

```
simulation_0.getSceneManager().createVectorScene("Vector
Scene", "Outline", "Vector");

Scene scene_2 =
  simulation_0.getSceneManager().getScene("Vector Scene 1");

scene_2.initializeAndWait();

fixedAspectAnnotationProp_0.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_0.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

FixedAspectAnnotationProp fixedAspectAnnotationProp_1 =
  ((FixedAspectAnnotationProp)
  scene_1.getAnnotationPropManager().getAnnotationProp("Logo
     "));

fixedAspectAnnotationProp_1.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_1.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

PartDisplayer partDisplayer_6 =
  ((PartDisplayer) scene_2.getCreatorDisplayer());

partDisplayer_6.initialize();
```

```
PartDisplayer partDisplayer_5 =
  ((PartDisplayer) scene_2.getDisplayerManager().getDisplayer
     ("Outline 1"));

partDisplayer_5.initialize();

VectorDisplayer vectorDisplayer_0 =
  ((VectorDisplayer) scene_2.getDisplayerManager().
     getDisplayer("Vector 1"));

vectorDisplayer_0.initialize();

scene_2.open(true);

CurrentView currentView_2 =
  scene_2.getCurrentView();

currentView_2.setInput(new DoubleVector(new double[]
{-5.2009992599487305, -7.169999927282333,
   -7.345000013709068}),
new DoubleVector(new double[] {-5.2009992599487305,
-7.169999927282333, 91.89512116659364}), new DoubleVector(new
double[] {0.0, 1.0, 0.0}), 25.906870266242805, 0);

vectorDisplayer_0.getParts().setObjects(boundary_3);

GlyphSettings glyphSettings_0 =
  vectorDisplayer_0.getGlyphSettings();

glyphSettings_0.setRelativeToModelLength(1.0);
```

```
scene_2.setPresentationName("Vector Velocity");

simulation_0.getSceneManager().createScalarScene("Scalar
Scene", "Outline", "Scalar");

Scene scene_3 =
  simulation_0.getSceneManager().getScene("Scalar Scene 1");

scene_3.initializeAndWait();

fixedAspectAnnotationProp_0.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_0.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_1.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_1.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

FixedAspectAnnotationProp fixedAspectAnnotationProp_2 =
  ((FixedAspectAnnotationProp)
  scene_2.getAnnotationPropManager().getAnnotationProp("Logo
    "));

fixedAspectAnnotationProp_2.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));
```

```
fixedAspectAnnotationProp_2.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

PartDisplayer partDisplayer_8 =
  ((PartDisplayer) scene_3.getCreatorDisplayer());

partDisplayer_8.initialize();

PartDisplayer partDisplayer_7 =
  ((PartDisplayer) scene_3.getDisplayerManager().getDisplayer
     ("Outline 1"));

partDisplayer_7.initialize();

ScalarDisplayer scalarDisplayer_1 =
  ((ScalarDisplayer) scene_3.getDisplayerManager().
     getDisplayer("Scalar 1"));

scalarDisplayer_1.initialize();

scene_3.open(true);

CurrentView currentView_3 =
  scene_3.getCurrentView();

currentView_3.setInput(new DoubleVector(new double[]
{-5.2009992599487305, -7.169999927282333,
   -7.345000013709068}),
new DoubleVector(new double[] {-5.2009992599487305,
    −7.169999927282333, 91.89512116659364}), new DoubleVector(new
    double[] {0.0, 1.0, 0.0}), 25.906870266242805, 0);
  }

  private void execute1() {

    Simulation simulation_0 =
      getActiveSimulation();

    Scene scene_3 =
      simulation_0.getSceneManager().getScene(”Scalar Scene 1”);

    ScalarDisplayer scalarDisplayer_1 =
      ((ScalarDisplayer) scene_3.getDisplayerManager().
         getDisplayer(”Scalar 1”));

    Region region_0 =
      simulation_0.getRegionManager().getRegion(”Region 1”);

    Boundary boundary_3 =
      region_0.getBoundaryManager().getBoundary(”Subtract.
         SymPlane”);

    scalarDisplayer_1.getParts().setObjects(boundary_3);

    PrimitiveFieldFunction primitiveFieldFunction_5 =
      ((PrimitiveFieldFunction)
      simulation_0.getFieldFunctionManager().getFunction(”
         Pressure”));
```

```
scalarDisplayer_1.getScalarDisplayQuantity()
.setFieldFunction(primitiveFieldFunction_5);

scalarDisplayer_1.setFillMode(1);

scene_3.setPresentationName("Scalar Pressure");

simulation_0.getSceneManager().createScalarScene("Scalar
Scene", "Outline", "Scalar");

Scene scene_4 =
  simulation_0.getSceneManager().getScene("Scalar Scene 1");

scene_4.initializeAndWait();

Scene scene_0 =
  simulation_0.getSceneManager().getScene("Geometry Scene 1")
     ;

FixedAspectAnnotationProp fixedAspectAnnotationProp_0 =
  ((FixedAspectAnnotationProp)
  scene_0.getAnnotationPropManager().getAnnotationProp("Logo
     "));

fixedAspectAnnotationProp_0.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_0.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));
```

```
Scene scene_1 =
  simulation_0.getSceneManager().getScene("scalar velocity");

FixedAspectAnnotationProp fixedAspectAnnotationProp_1 =
  ((FixedAspectAnnotationProp)
  scene_1.getAnnotationPropManager().getAnnotationProp("Logo
     "));

fixedAspectAnnotationProp_1.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_1.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

Scene scene_2 =
  simulation_0.getSceneManager().getScene("Vector Velocity");

FixedAspectAnnotationProp fixedAspectAnnotationProp_2 =
  ((FixedAspectAnnotationProp)
  scene_2.getAnnotationPropManager().getAnnotationProp("Logo
     "));

fixedAspectAnnotationProp_2.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_2.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

FixedAspectAnnotationProp fixedAspectAnnotationProp_3 =
  ((FixedAspectAnnotationProp)
  scene_3.getAnnotationPropManager().getAnnotationProp("Logo
     "));

fixedAspectAnnotationProp_3.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

fixedAspectAnnotationProp_3.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

PartDisplayer partDisplayer_10 =
  ((PartDisplayer) scene_4.getCreatorDisplayer());

partDisplayer_10.initialize();

PartDisplayer partDisplayer_9 =
  ((PartDisplayer) scene_4.getDisplayerManager().getDisplayer
     ("Outline 1"));

partDisplayer_9.initialize();

ScalarDisplayer scalarDisplayer_2 =
  ((ScalarDisplayer) scene_4.getDisplayerManager().
     getDisplayer("Scalar 1"));

scalarDisplayer_2.initialize();

scene_4.open(true);

CurrentView currentView_4 =
  scene_4.getCurrentView();
```

```
currentView_4.setInput(new DoubleVector(new double[]
{-5.2009992599487305, -7.169999927282333,
   -7.345000013709068}),
new DoubleVector(new double[] {-5.2009992599487305,
-7.169999927282333, 91.89512116659364}), new DoubleVector(new
double[] {0.0, 1.0, 0.0}), 25.906870266242805, 0);

Boundary boundary_0 =
  region_0.getBoundaryManager().getBoundary("Subtract.Body");

scalarDisplayer_2.getParts().setObjects(boundary_0);

PrimitiveFieldFunction primitiveFieldFunction_6 =
  ((PrimitiveFieldFunction)
  simulation_0.getFieldFunctionManager().getFunction("
     WallShearStress"));

VectorMagnitudeFieldFunction vectorMagnitudeFieldFunction_1 =
  ((VectorMagnitudeFieldFunction)
  primitiveFieldFunction_6.getMagnitudeFunction());

scalarDisplayer_2.getScalarDisplayQuantity()
.setFieldFunction(vectorMagnitudeFieldFunction_1);

scalarDisplayer_2.setFillMode(1);

scene_4.setPresentationName("Wall shear stress");

Units units_0 =
   simulation_0.getUnitsManager().getPreferredUnits(new
   IntVector(new int[] {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}));

scene_4.setTransparencyOverrideMode(1);

scene_4.getCreatorGroup().setObjects(region_0);

currentView_4.setInput(new DoubleVector(new double[]
{-4.095828551434292, -5.452952733652851,
    -3.0522336421437544}),
new DoubleVector(new double[] {50.815338286573414,
79.85971814310008, -8.341137674763642}), new DoubleVector(new
double[] {-0.4787973154414936, 0.2561171043871874,
-0.8397363631321095}), 26.521509765591478, 0);

currentView_4.setInput(new DoubleVector(new double[]
{-21.336521410525393, -20.616327417378912,
    2.253827145854747}),
new DoubleVector(new double[] {74.51765863987015,
7.0427587008479104, -45.013784127639234}), new DoubleVector(
    new
double[] {-0.48540715249204747, 0.2510015897360899,
-0.8374831928221116}), 28.819138012913754, 0);

currentView_4.setInput(new DoubleVector(new double[]
{-16.001498082390274, -18.875508500201136,
0.38052651337301746}), new DoubleVector(new double[]
{88.13956696658751, 25.624610246003243, 3.381199462357289}),
new DoubleVector(new double[] {-0.13477714553603723,
0.3771961686635576, -0.9162740700176399}),
    29.574626863891524,
0);

PlaneSection planeSection_0 =
   (PlaneSection)
   simulation_0.getPartManager().createImplicitPart(new
   NeoObjectVector(new Object[] {region_0}), new
   DoubleVector(new double[] {0.0, 0.0, 1.0}), new
   DoubleVector(new double[] {-4.525, -6.96, 0.154}), 0, 1,
      new
   DoubleVector(new double[] {0.0}));

LabCoordinateSystem labCoordinateSystem_0 =
   simulation_0.getCoordinateSystemManager().
      getLabCoordinateSystem();

planeSection_0.setCoordinateSystem(labCoordinateSystem_0);

Coordinate coordinate_6 =
   planeSection_0.getOriginCoordinate();

coordinate_6.setCoordinate(units_0, units_0, units_0, new
DoubleVector(new double[] {-4.525, -6.96, 0.154}));

Coordinate coordinate_7 =
   planeSection_0.getOrientationCoordinate();

coordinate_7.setCoordinate(units_0, units_0, units_0, new
DoubleVector(new double[] {0.0, 0.0, 1.0}));
```

```
coordinate_7.setValue(new DoubleVector(new double[] {0.0,
    0.0, 1.0}));

coordinate_6.setValue(new DoubleVector(new double[] {-4.525,
    -6.96, 0.154}));

SingleValue singleValue_0 =
   planeSection_0.getSingleValue();

singleValue_0.getValueQuantity().setValue(0.0);

singleValue_0.getValueQuantity().setUnits(units_0);

RangeMultiValue rangeMultiValue_0 =
   planeSection_0.getRangeMultiValue();

rangeMultiValue_0.setNValues(2);

rangeMultiValue_0.getStartQuantity().setValue(0.0);

rangeMultiValue_0.getStartQuantity().setUnits(units_0);

rangeMultiValue_0.getEndQuantity().setValue(1.0);

rangeMultiValue_0.getEndQuantity().setUnits(units_0);

DeltaMultiValue deltaMultiValue_0 =
   planeSection_0.getDeltaMultiValue();
```

```
deltaMultiValue_0.setNValues(2);

deltaMultiValue_0.getStartQuantity().setValue(0.0);

deltaMultiValue_0.getStartQuantity().setUnits(units_0);

deltaMultiValue_0.getDeltaQuantity().setValue(1.0);

deltaMultiValue_0.getDeltaQuantity().setUnits(units_0);

MultiValue multiValue_0 =
  planeSection_0.getArbitraryMultiValue();

multiValue_0.getValueQuantities().setUnits(units_0);

multiValue_0.getValueQuantities().setArray(new DoubleVector(
   new double[] {0.0}));

planeSection_0.setValueMode(0);

scene_4.setTransparencyOverrideMode(0);

scene_4.setTransparencyOverrideMode(1);

scene_4.setTransparencyOverrideMode(0);

simulation_0.getSceneManager().createScalarScene("Scalar
Scene", "Outline", "Scalar");

Scene scene_5 =
  simulation_0.getSceneManager().getScene("Scalar Scene 1");

scene_5.initializeAndWait();

fixedAspectAnnotationProp_0.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

  FixedAspectAnnotationProp fixedAspectAnnotationProp_4 =
  ((FixedAspectAnnotationProp)
  scene_4.getAnnotationPropManager().getAnnotationProp("Logo
     "));

fixedAspectAnnotationProp_4.setPosition(new DoubleVector(new
double[] {0.015, 0.85, 0.0}));

PartDisplayer partDisplayer_12 =
  ((PartDisplayer) scene_5.getCreatorDisplayer());

partDisplayer_12.initialize();

PartDisplayer partDisplayer_11 =
  ((PartDisplayer) scene_5.getDisplayerManager().getDisplayer
     ("Outline 1"));

partDisplayer_11.initialize();

ScalarDisplayer scalarDisplayer_3 =
  ((ScalarDisplayer) scene_5.getDisplayerManager().
     getDisplayer("Scalar 1"));
```

```
scalarDisplayer_3.initialize();

scene_5.open(true);

CurrentView currentView_5 =
  scene_5.getCurrentView();

currentView_5.setInput(new DoubleVector(new double[]
{-5.2009992599487305, -7.169999927282333,
   -7.345000013709068}),
new DoubleVector(new double[] {-5.2009992599487305,
-7.169999927282333, 91.89512116659364}), new DoubleVector(new
double[] {0.0, 1.0, 0.0}), 25.906870266242805, 0);

scalarDisplayer_3.getParts().setObjects(planeSection_0);

PrimitiveFieldFunction primitiveFieldFunction_4 =
  ((PrimitiveFieldFunction)
  simulation_0.getFieldFunctionManager().getFunction("
     Velocity"));

VectorMagnitudeFieldFunction vectorMagnitudeFieldFunction_0 =
  ((VectorMagnitudeFieldFunction)
  primitiveFieldFunction_4.getMagnitudeFunction());

scalarDisplayer_3.getScalarDisplayQuantity()
.setFieldFunction(vectorMagnitudeFieldFunction_0);

scalarDisplayer_3.setFillMode(1);
```

```
scene_5.setPresentationName("cross plane velocity");

ResidualPlot residualPlot_0 =
  ((ResidualPlot) simulation_0.getPlotManager().getPlot("
     Residuals"));

residualPlot_0.open();

simulation_0.getSimulationIterator().run();

FrontalAreaReport frontalAreaReport_0 =
  ((FrontalAreaReport)
  simulation_0.getReportManager().getReport("Frontal Area 1")
     );

frontalAreaReport_0.printReport();

MaxReport maxReport_1 =
  ((MaxReport) simulation_0.getReportManager().getReport("
     average Cl"));

maxReport_1.printReport();

MaxReport maxReport_0 =
  ((MaxReport) simulation_0.getReportManager().getReport("
     average Cd"));

maxReport_0.printReport();

    String filename =
                simulation_0.getPresentationName();

          simulation_0.saveState(resolvePath(filename+"_ran.sim"));
    }
  }
```

APPENDIX B. METHOD FOR CALCULATING ERROR IN CFD SIMULATIONS

This section explains the error calculations mentioned in Chapter 2. The steps are taken from the GCI article in [11]. Table A.1 below shows the three simulations used for GCI calculation, and the corresponding C_D for fully laminar flow. The baseline vehicle simulations were used for the GCI calculations, and the results were extended to the Model C simulations since the grids are similar.

Table B.1: Grid Sizes used for Laminar GCI Calculation

Simulation	**Number of Cells**	C_D
f_1	8,064,318	0.0683
f_2	4,021,998	0.0700
f_3	2,024,576	0.0731

The first step is to determine the order of convergence, p, with the equation

$$p = ln(\frac{f_3 - f_2}{f_2 - f_1})/ln(r)$$

where r = the grid refinement ratio, which in this case is 2, referring to each grid size being approximately twice as large as the next one. For this case $p = 0.902$. The order of convergence should approach $r = 2$, and since there is a difference in this case we can conclude that there are some non-linearities in the solution or some stretching of the grid in some locations.

To calculate GCI, we use the equation

$$GCI_{12} = \frac{1.25|f_1 - f_2|/f_1}{2^p - 1}$$

For this case, we get $GCI_{12} = 0.035$, which is converted to a percentage of 3.5%. For GCI_{23}, we get 0.064, or 6.4%. These percentages are the error bands that are used to approximate the error in the CFD results. GCI_{12} is used since it represents the finer mesh GCI. Another check is to make sure the value of C_D obtained falls in the asymptotic range of convergence. This can be checked by the equation

$$C = \frac{GCI_{23} * 100}{r^p * GCI_{12} * 100}$$

where C should be close to 1. Using the calculted values of GCI_{12} and GCI_{23}, we get C = 0.976, which is close to 1.

The same calculations can be extended to the turbulent results. Table A.2 shows the simulations f_1, f_2, f_3, and the corresponding turbulent C_D.

Table B.2: Grid Sizes used for Turbulent GCI Calculation

Simulation	**Number of Cells**	C_D
f_1	8,064,318	0.1072
f_2	4,021,998	0.1075
f_3	2,024,576	0.10794

For the turbulent case, $p = 0.554$, which yields $GCI_{12} = 0.007$ and $GCI_{23} = 0.0109$. Checking the asymptotic range of convergence yields $C = 0.997$, which is close to 1. Thus the turbulent C_D results have an error band of 0.7%.