Washington University in St. Louis

Washington University Open Scholarship

All Theses and Dissertations (ETDs)

January 2009

Numerical Drag Reduction Studies on Generic Truck

Miles Bellman
Washington University in St. Louis

Follow this and additional works at: https://openscholarship.wustl.edu/etd

Recommended Citation

Bellman, Miles, "Numerical Drag Reduction Studies on Generic Truck" (2009). *All Theses and Dissertations (ETDs)*. 522.
https://openscholarship.wustl.edu/etd/522

This Thesis is brought to you for free and open access by Washington University Open Scholarship. It has been accepted for inclusion in All Theses and Dissertations (ETDs) by an authorized administrator of Washington University Open Scholarship. For more information, please contact digital@wumail.wustl.edu.

WASHINGTON UNIVERSITY IN ST. LOUIS
School of Engineering and Applied Science
Department of Mechanical, Aerospace, and Structural Engineering

Thesis Examination Committee:
Ramesh K. Agarwal, Chair
Kenneth L. Jerina
David A. Peters

NUMERICAL DRAG REDUCTION STUDIES ON GENERIC TRUCK MODELS USING PASSIVE AND ACTIVE FLOW CONTROL

by

Miles E. Bellman

A thesis presented to the School of Engineering
of Washington University in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

August 2009
Saint Louis, Missouri

ABSTRACT OF THE THESIS

Numerical Drag Reduction Studies on Generic Truck Models Using Passive and Active Flow Control

by

Miles E. Bellman

Master of Science in Mechanical Engineering

Washington University in St. Louis, 2009

Research Advisor: Professor Ramesh K. Agarwal

Drag reduction studies on generic truck models using passive shape optimization and active flow control are studied using numerical methods. A genetic algorithm is used to find optimal truck front shapes for reduced drag. The best shape found gives a 3.4% drag reduction. Active flow control is used to reduce drag by using oscillatory synthetic jet actuators on a generic truck body, a D-shaped bluff body, and the Ahmed body. Experimental data is available for these three configurations without and with active flow control. Actuators at the back of the trucks energize separating boundary layers and delay the shedding of low pressure-inducing vortices in the wake resulting in drag reduction. A maximum drag reduction of 16% and 21% was found to be possible for the two-dimensional model of a generic truck and a two-dimensional D-shaped bluff body respectively; these results are in close agreement with the experimental data. Active flow control with steady blowing on the three-dimensional Ahmed body however did not show drag reduction. This calculation requires further investigation.

Contents

List of Tables v

List of Figures vi

Acknowledgements xii

Chapter 1 Introduction 1
- 1.1 Motivation 1
- 1.2 Current and Future Forecast of Fuel Consumption by Ground Vehicles 2
- 1.3 Improving Fuel Efficiency of Ground Vehicles by Reducing the Aerodynamic Drag 3
- 1.4 Scope of the Thesis 6
  - 1.4.1 Shape Optimization Using a Genetic Algorithm 6
  - 1.4.2 Active Flow Control Using Synthetic Jet Actuators 7

Chapter 2 Literature Survey 8
- 2.1 Large Trucks Drag Reduction Using Active Flow Control 9
- 2.2 Feedback Shear Layer Control for Bluff Body Drag Reduction 10
- 2.3 Application of Slope-Seeking to a Generic Car Model for Active Drag Control 12
- 2.4 Drag Reduction of a Generic Car Model Using Steady Blowing 14

Chapter 3 Computational Fluid Dynamics (CFD) Solver 16
- 3.1 GAMBIT 16
- 3.2 FLUENT 19

Chapter 4 Passive Flow Control by Shape Optimization Using a Genetic Algorithm 21
- 4.1 Introduction 21
- 4.2 Genetic Algorithms 22
- 4.3 Airfoil and Truck Parameterization 24
  - 4.3.1 Joukowski Transformation for Airfoil Parameterization 24
  - 4.3.2 Bézier Curves for Truck Front Parameterization 25
- 4.4 Algorithm Implementation 26
  - 4.4.1 Airfoil Constraints 28
  - 4.4.2 Truck Constraints 28
- 4.5 Mesh Generation 29
- 4.6 Flow Field Computation 33
- 4.7 Results and Discussion 34
  - 4.7.1 Airfoil Optimization 34
  - 4.7.2 Truck Front-Shape Optimization 39
- 4.8 Conclusions 47

Chapter 5 Active Flow Control Using Synthetic Jet Actuators (SJA) 49
- 5.1 Introduction 49
- 5.2 Bluff Body Wakes 50
- 5.3 Synthetic Jet Actuators (SJAs) 52

5.4 Generic Truck Body 56
  5.4.1 Introduction 56
  5.4.2 Computational Solution Procedure....................57
  5.4.3 Results and Analysis....................61
5.5 D-shaped Bluff Body....................77
  5.5.1 Introduction 77
  5.5.2 Computational Solution Procedure....................78
  5.5.3 Results and Analysis....................81
5.6 Ahmed Body....................87
  5.6.1 Introduction 87
  5.6.2 Computational Solution Procedure....................87
  5.6.3 Results and Analysis....................90
5.7 Conclusions 95
Chapter 6 Future Work 98

Appendix A Java Code for Airfoil Shape Optimization Using a Genetic Algorithm 100

Appendix B Java Code for Truck Front Shape Optimization Using a Genetic Algorithm 130

Appendix C MATLAB Postprocessing Codes 150

Appendix D User Defined Function (UDF) for Synthetic Jet Actuators in FLUENT 153

References....................154

Vita....................156

List of Tables

Table 4.1 Results for C_l- optimized airfoils, Re_c = 100,000 34

Table 4.2 Results for C_d-optimized truck front 39

Table 5.1 Experimental versus computed C_d for the generic truck body for the baseline case (without AFC) 62

Table 5.2 Computed active flow control results for the generic truck body with two SJAs in configuration 1 (Figure 5.11) 63

Table 5.3 Computed active flow control results for the generic truck body with two SJAs in configuration 2 (Figure 5.12) 64

Table 5.4 Computed active flow control results for the generic truck body with three SJAs in configuration 3 (Figure 5.13) 65

Table 5.5 Experimental versus computed C_d for the D-shaped bluff body 82

Table 5.6 Experimental versus computed drag coefficient C_d for the Ahmed body 91

List of Figures

Figure 1.1 Fuel consumption by vehicle category in U.S. [1] 2

Figure 1.2 Actual and predicted fuel consumption by vehicle category [1] 3

Figure 1.3 Sources of fuel efficiency losses in a heavy truck [1] 4

Figure 1.4 Three factors contributing to fuel consumption due to changes in aerodynamic drag [1] 5

Figure 1.5 Means of achieving drag reduction [1] 7

Figure 2.1 Cross section of an active flow control device or actuator employing suction and oscillatory blowing [6] 9

Figure 2.2 Generic truck body [6] 10

Figure 2.3 D-shaped bluff body [7] 11

Figure 2.4 Three-dimensional vertical structures behind the Ahmed body [8] 13

Figure 2.5 Steady and periodic actuators placed on the Ahmed body for active flow control [8] 13

Figure 2.6 Three-dimensional multi-block structured grid around the Ahmed body [10] 14

Figure 2.7 Steady blowing actuators on the edges of the slant and vertical surfaces on the backside of the Ahmed body [10] 15

Figure 3.1 Quadrilateral structured grid 17

Figure 3.2 Triangular unstructured grid 18

Figure 3.3 Quadrilateral unstructured grid 18

Figure 4.1 Individuals with n alleles 22

Figure 4.2 General Crossover 23

Figure 4.3 Mutation 23

Figure 4.4 Joukowski Transformation 25

Figure 4.5 Bézier curve cubic parameterization 26

Figure 4.6 Illustration of information flow in genetic algorithm implementation process 26

Figure 4.7 Extrapolation-based crossover 27

Figure 4.8 Automatically generated adaptive mesh for airfoil 30
Figure 4.9 Automatically generated adaptive mesh for airfoil, zoomed-in view 30
Figure 4.10 Automatically generated adaptive mesh for airfoil, zoomed-in view 31
Figure 4.11 Automatically generated adaptive mesh for truck 32
Figure 4.12 Automatically generated adaptive mesh for truck, zoomed-in view 32
Figure 4.13 Automatically generated adaptive mesh for truck, zoomed-in front view ... 33
Figure 4.14 Velocity vectors for the C_l-optimized airfoil at $a = 0°$, $Re_c = 100{,}000$ 35
Figure 4.15 Pressure coefficient plot for the C_l-optimized airfoil at $a = 0°$, $Re_c = 100{,}000$ 36
Figure 4.16 Convergence plot for the C_l-optimized airfoil at $a = 0°$, $Re_c = 100{,}000$ 36
Figure 4.17 Velocity vectors for the C_l-optimized airfoil at $a = 2°$, $Re_c = 100{,}000$ 37
Figure 4.18 Pressure coefficient plot for the C_l-optimized airfoil at $a = 2°$, $Re_c = 100{,}000$ 38
Figure 4.19 Convergence plot for the C_l-optimized airfoil at $a = 2°$, $Re_c = 100{,}000$ 38
Figure 4.20 Convergence plot for Case 1 C_d-optimized truck front, $Re_H = 303{,}957$ 40
Figure 4.21 Velocity contours for Case 1 C_d-optimized truck front, $Re_H = 303{,}957$ 40
Figure 4.22 Convergence plot for Case 2 C_d-optimized truck front, $Re_H = 303{,}957$ 41
Figure 4.23 Velocity contours for the Case 2 C_d-optimized truck front, $Re_H = 303{,}957$ 42
Figure 4.24 Convergence plot for Case 3 C_d-optimized truck front, $Re_H = 303{,}957$ 43
Figure 4.25 Velocity contours for the Case 3 C_d-optimized truck front, $Re_H = 303{,}957$ 43
Figure 4.26 Velocity contours for Case 4 C_d-optimized truck front, $Re_H = 303{,}957$ 44
Figure 4.27 Convergence plot for Case 4 C_d-optimized truck front, $Re_H = 303{,}957$ 45
Figure 4.28 Velocity contours for Case 5 C_d-optimized truck front, $Re_H = 303{,}957$ 46
Figure 4.29 Convergence plot for Case 5 C_d-optimized truck front, $Re_H = 303{,}957$ 46
Figure 5.1 Kármán vortex street behind a cylinder [20] 50
Figure 5.2 Asymmetrical vortex shedding from a D-shaped bluff body [7] 51
Figure 5.3 Symmetrical vortex shedding behind the D-shaped bluff body as a result of active flow control [7] 52
Figure 5.4 Schematic of SJA operation [21] 53
Figure 5.5 Detailed schematic of the operation of the SJA [21] 55

Figure 5.6 Typical parts of the SJA (on left) and SJA assembly (on right) [21]................ 56
Figure 5.7 Generic truck dimensions [5]... 57
Figure 5.8 Generic truck body: complete computational domain with structured mesh 58
Figure 5.9 Generic truck body with zoomed-in mesh near the body................................ 58
Figure 5.10 Generic truck body with zoomed-in adapted mesh very close to the body and synthetic jet regions 59
Figure 5.11 Configuration 1: Two SJAs at the rear face of the truck in the direction of the free stream.. 60
Figure 5.12 Configuration 2: Two SJAs at the rear face of the truck at an inclination of 45° to the free stream.. 60
Figure 5.13 Configuration 3: Two SJAs at the rear face of the truck in the direction of the free stream and one SJA on the top surface of the truck........................ 60
Figure 5.14 Experimental C_d versus computed C_d for the generic truck body for the baseline case (without AFC).. 65
Figure 5.15 Computed C_d with active flow control for the generic truck body for three SJA configurations: config. 1 (Figure 5.11), config. 2 (Figure 5.12), and config. 3 (Figure 5.13) 66
Figure 5.16 Computed velocity contours for the baseline flow (without AFC) at $Re = 3.08 \times 10^5$ for the generic truck body.. 67
Figure 5.17 Computed velocity contours for the baseline flow (without AFC) at $Re = 6.16 \times 10^5$ for the generic truck body.. 67
Figure 5.18 Computed velocity contours for the baseline flow (without AFC) at $Re = 9.24 \times 10^5$ for the generic truck body.. 68
Figure 5.19 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at $Re = 3.08 \times 10^5$.. 68
Figure 5.20 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at $Re = 3.08 \times 10^5$.............. 69
Figure 5.21 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at $Re = 3.08 \times 10^5$.. 69

Figure 5.22 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at Re = 3.08 x 10^5.............. 70

Figure 5.23 Computed velocity contours for flow with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at Re = 3.08 x 10^5 70

Figure 5.24 Baseline C_p versus C_p with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at Re = 3.08 x 10^5 71

Figure 5.25 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at Re = 6.16 x 10^5.. 71

Figure 5.26 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at Re = 6.16 x 10^5.............. 72

Figure 5.27 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at Re = 6.16 x 10^5.. 72

Figure 5.28 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at Re = 6.16 x 10^5.............. 73

Figure 5.29 Computed velocity contours for flow with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at Re = 6.16 x 10^5 73

Figure 5.30 Baseline C_p versus C_p with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at Re = 6.16 x 10^5 74

Figure 5.31 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at Re = 9.24 x 10^5.. 74

Figure 5.32 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at Re = 9.24 x 10^5.............. 75

Figure 5.33 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at Re = 9.24 x 10^5.. 75

Figure 5.34 Baseline C_p versus C_p with active flow control C_p for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at Re = 9.24 x 10^5..... 76

Figure 5.35 Computed velocity contours for flow with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at *Re* = 9.24 x 10^5 76

Figure 5.36 Baseline C_p versus C_p with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at *Re* = 9.24 x 10^5 77

Figure 5.37 Geometry of the D-shaped bluff body [7] 79

Figure 5.38 D-shaped bluff body: complete computational domain with structured mesh 79

Figure 5.39 D-shaped bluff body with zoomed-in mesh near the body 80

Figure 5.40 D-shaped bluff body with zoomed-in adapted mesh very close to the body and synthetic jet regions 80

Figure 5.41 Two SJA locations and their exit angles on the rear face of the D-shaped bluff body 81

Figure 5.42 Experimental versus computed C_d for the D-shaped bluff body 83

Figure 5.43 Computed velocity contours for the D-shaped bluff body for the baseline case (without AFC) at *Re* = 35,000 84

Figure 5.44 Computed velocity contours for the D-shaped bluff body for the baseline case (without AFC) at *Re* = 70,000 84

Figure 5.45 Computed velocity contours for the D-shaped bluff body with AFC at *Re* = 35,000 85

Figure 5.46 Computed C_p for the D-shaped bluff body for the baseline case (without AFC) and the case with AFC at *Re* = 35,000 85

Figure 5.47 Computed velocity contours for the D-shaped bluff body with AFC at *Re* = 70,000 86

Figure 5.48 Computed C_p for the D-shaped bluff body for the baseline case (without AFC) and the case with AFC at *Re* = 70,000 86

Figure 5.49 Ahmed body geometry (all dimensions are in mm) 88

Figure 5.50 3-D structured mesh surrounding the Ahmed body 88

Figure 5.51 Surface mesh on the Ahmed body (view showing the details of the front part) 89

Figure 5.52 Surface mesh on the Ahmed body (view showing the details of the back part) 89

Figure 5.53 Spanwise vortices behind the Ahmed body.. 91

Figure 5.54 Longitudinal vertical structures behind the Ahmed body 92

Figure 5.55 Isometric view of computed C_p contours for baseline flow (without AFC) past the Ahmed body, Re = 500,000... 93

Figure 5.56 Computed C_p contours on the front face of the Ahmed body for baseline flow (without AFC), Re = 500,000.. 93

Figure 5.57 Computed C_p contours on the rear face of the Ahmed body for the baseline flow (without AFC), Re = 500,000... 94

Figure 5.58 Isometric view of computed C_p contours on the Ahmed body for AFC with steady blowing, Re = 500,000.. 94

Figure 5.59 Computed C_p contours on the rear face of the Ahmed body for AFC with steady blowing, Re = 500,000.. 95

Acknowledgements

Thanks to my parents for their support and to Professor Ramesh Agarwal for believing I could do this.

Miles E. Bellman

Washington University in St. Louis
August 2009

Chapter 1

Introduction

1.1 Motivation

The world needs sources of energy to power homes, businesses, and transportation vehicles. Much of this energy comes from fossil fuels such as oil, coal, and natural gas. In particular, most of the energy for all forms of transportation comes from oil. With the continuous increase in population as well as the large growth in the transportation sector, there is an increasing demand for oil worldwide. The rising demand coupled with dwindling supplies of oil (oil being a non-renewable energy source) has resulted in large increases in the price of oil in recent years. In addition, the consumption of oil is also a major contributor to greenhouse gas emmisions, influencing climate change.

Thus, the world is currently facing the urgency of tackling the two somewhat coupled problems: (1) how to reduce the use of non-renewable fossil fuels by improving the energy efficiency via technological innovation and (2) developing alternate clean sources of renewable energy (e.g. wind, solar, geothermal, etc.). In this thesis, we address the problem of improving the energy efficiency of ground vehicles by technological innovation. In particular, we investigate the possibility of reducing the aerodynamic drag of ground vehicles by shape optimization and by flow modification using recently developed active flow control (AFC) devices. A 15% reduction in aerodynamic drag results in approximately 5% in fuel savings.

1.2 Current and Future Forecast of Fuel Consumption by Ground Vehicles

Based on the 2004 data, ground vehicles (automobiles, light trucks, and other heavy duty vehicles) consume 77% of all oil used in all modes of transportation in the U.S. as shown in Figure 1.1 [1]. Currently (2008), about 12 million barrels per day of oil are used by ground vehicles as shown in Figure 1.2 [1]. This figure will rise to approximately 16 million barrels per day by 2025. It is easily estimated that a 1% increase in fuel economy for the ground vehicles will save about 245 million gallons of fuel each year. There are currently 700 million ground vehicles worldwide; this number is expected to rise to 2 billion by 2025. In addition, current ground vehicles contribute 12% to greenhouse gas emmisions worldwide. This number is expected to increase to 16% by 2025 if business is continued as usual. It turns out that at least a 5% increase in fuel economy can be achieved by retrofitting the proven technologies of active flow control on old/new vehicles. On new vehicles, additional gains in fuel economy can be achieved by shape optimization.

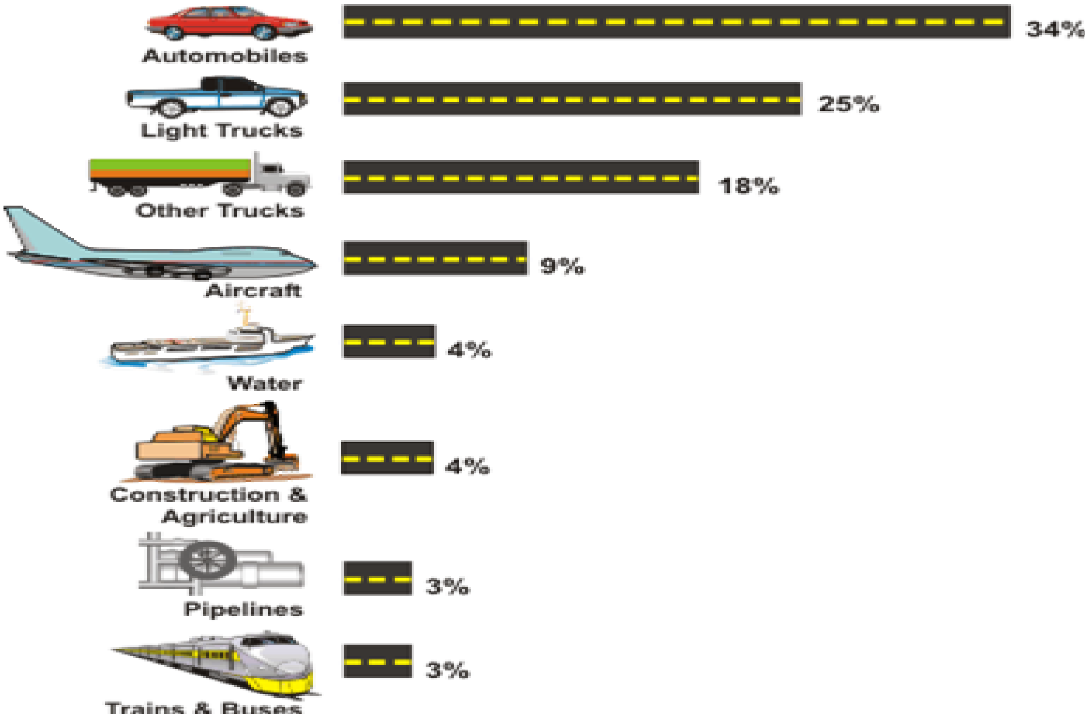
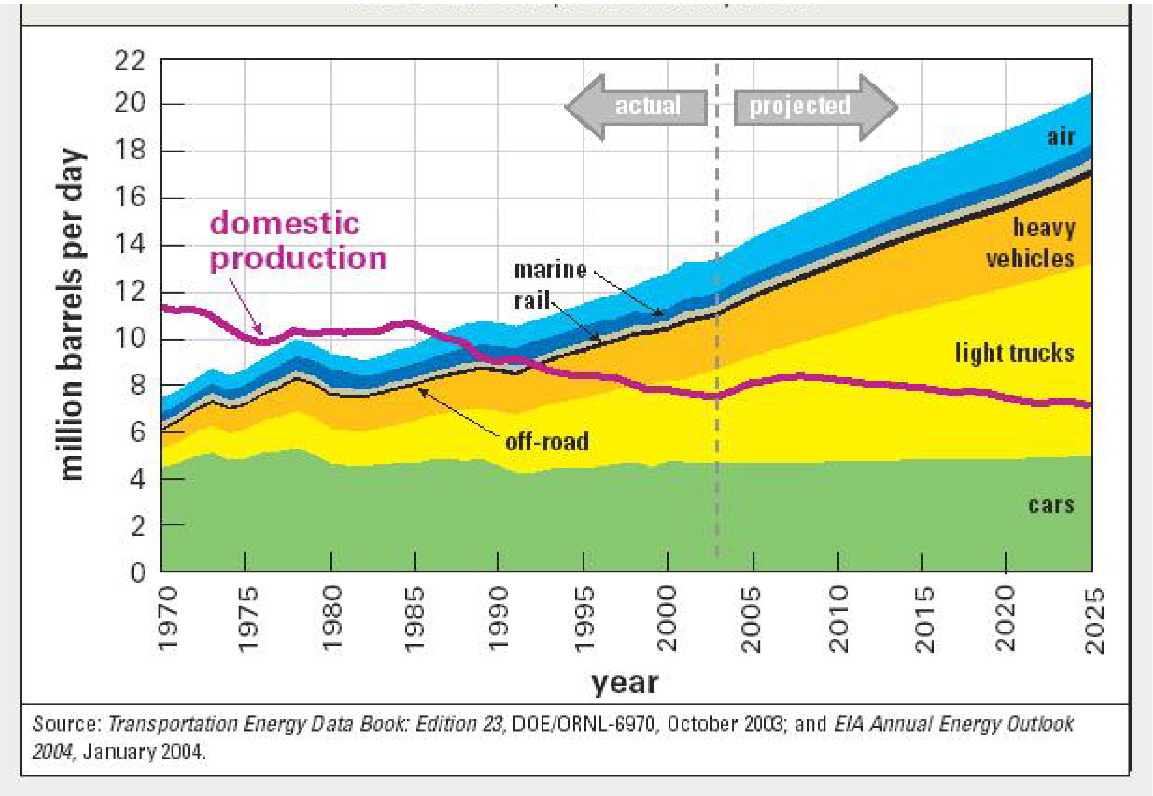


Figure 1.1 Fuel consumption by vehicle category in U.S. [1]



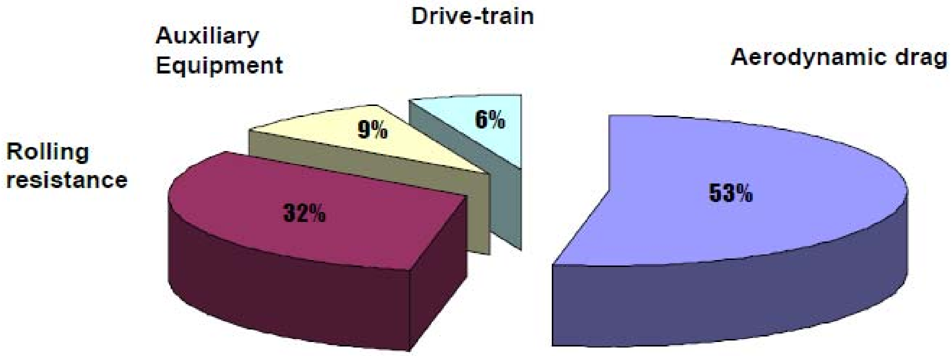
Source: *Transportation Energy Data Book: Edition 23*, DOE/ORNL-6970, October 2003; and *EIA Annual Energy Outlook 2004*, January 2004.

Figure 1.2 Actual and predicted fuel consumption by vehicle category [1]

1.3 Improving Fuel Efficiency of Ground Vehicles by Reducing the Aerodynamic Drag

With the sharp increase in gas prices during the summer of 2008 and the increasing calls for reductions of greenhouse gas emissions, the Obama administration has set new fuel efficiency standards for cars and light trucks [2]. Similarly, there is a need for improvement in the fuel efficiency of heavy shipping trucks. The goal of this thesis is to study both passive shape optimization and active flow control methods for improving the fuel efficiency of heavy trucks by reducing their aerodynamic drag.

Fuel efficiency is generally defined as the proportion of energy released by a fuel combustion process which is converted into useful work [3]. Heavy trucks use fuel to overcome rolling resistance and aerodynamic drag, and for running the drive train and auxialliary equipment as shown in Figure 1.3 [1]. As indicated in this figure, it is possible to reduce energy losses in all four categories by employing presently available technology.



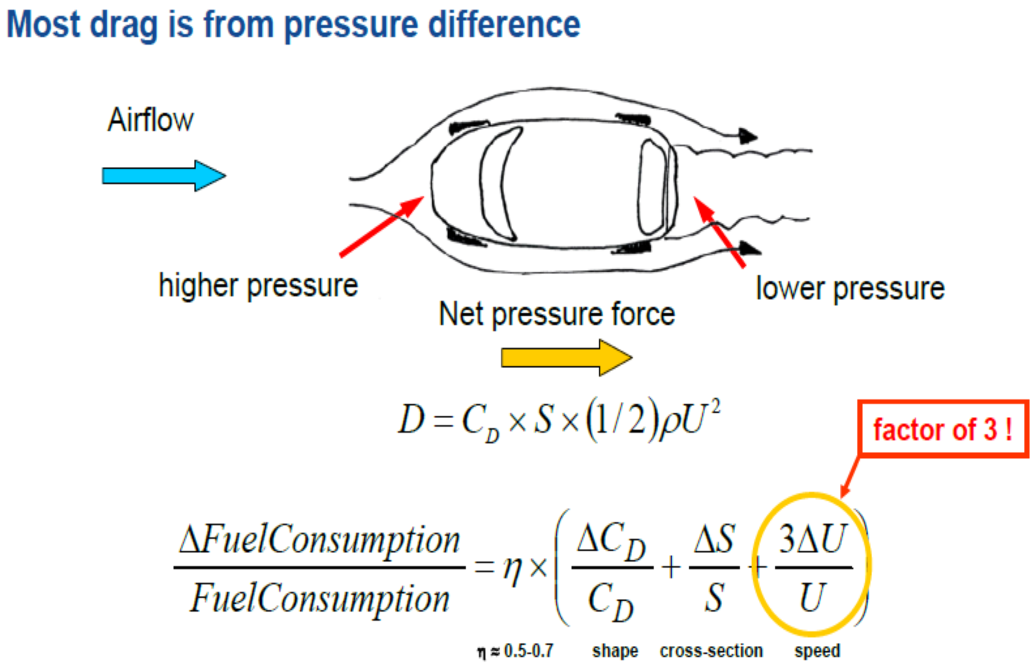
Losses in nearly all of these categories can be reduced by employing presently available technology

Figure 1.3 Sources of fuel efficiency losses in a heavy truck [1]

Six percent of the energy losses come from the drive train or from all the power transmitting components between the engine and the wheels. Nine percent of the losses come from the powering of auxiliary equipment like the air conditioning, power steering, and windshield wipers. Thirty-two percent of the losses come from resistance between the wheels and the road. The largest loss, about fifty-three percent, comes from overcoming the aerodynamic drag.

Drag is defined as the force that opposes the relative motion of an object through a fluid. It consists of two components: skin friction drag due to shear stress on the truck walls, and the aerodynamic drag due to the difference in pressure between the front and back sides of a vehicle. Aerodynamic drag is a much larger contributor to the overall drag compared to the skin friction drag. Figure 1.4 [1] shows the three factors that contribute to the aerodynamic drag: the shape of the vehicle, the cross-sectional area perpendicular to the free-stream, and the speed of the vehicle. It should be noted from

Figure 1.4 that the change in speed of a vehicle contributes to the change in fuel requirements by a factor of three.



Most drag is from pressure difference
Airflow
higher pressure
Net pressure force
lower pressure
$D = C_D \times S \times (1/2)\rho U^2$
factor of 3 !
$\frac{\Delta FuelConsumption}{FuelConsumption} = \eta \times \left(\frac{\Delta C_D}{C_D} + \frac{\Delta S}{S} + \frac{3\Delta U}{U} \right)$
η ≈ 0.5-0.7
shape
cross-section
speed

Figure 1.4 Three factors contributing to fuel consumption due to changes in aerodynamic drag [1]

Figure 1.4 also shows the schematic of pressure difference on a vehicle that contributes to aerodynamic drag. Neglecting the viscous effects, the inviscid steady state Bernoulli's equation (1) can mathematically describe the pressure drag. Because the conditions are steady and the air can be considered incompressible, Bernoulli's equation along a streamline can be expressed as:

$$\frac{P}{\rho} + \frac{V^2}{2} + gz = const. \tag{1}$$

where *P*, *ρ*, *V*, and *z* are the pressure, density, velocity, and elevation of a fluid particle along the streamline respectively. The acceleration of gravity is *g*. As the vehicle moves through air, the oncoming air hits the front of the vehicle and essentially becomes stagnant. Thus, in Bernoulli's equation (1), *V*≈0 and *z* can be taken as zero because the change in elevation of the oncoming fluid can be neglected. Therefore, P becomes large near the truck front. At the back of the truck, the boundary layers attached to the truck body separate and form a vortex street behind the base of the truck. The recirculating flow region has a larger velocity than the front of the truck and thus lower pressure. It should be noted that the recirculating flow behind the truck occurs due to viscous effects and the Bernoulli equation is not applicable. It can be used to qualitatively explain the lower pressure in the base region, but cannot be used to predict the pressure.

If the average pressure on the front and back part of the control volume surrounding the truck could be determined computationally or experimentally, the drag force due to the pressure difference would be given by:

$$F_D = Area\left(P_{front} - P_{back}\right), \tag{2}$$

where *Area* is the front area of the truck in the control volume. This is also equivalent to the back area in the control volume. Therefore, from equation (2), the aerodynamic drag due to pressure difference can be decreased by lowering the P_{front} by shape optimization and increasing the P_{back} by active flow control.

1.4 Scope of the Thesis

Figure 1.5 shows various regions of flow that contribute to aerodynamic drag of a heavy duty truck. In this thesis, the focus is on reducing the pressure near the truck front by shape optimization and increasing the back pressure with active flow control so that the aerodynamic pressure drag is reduced.

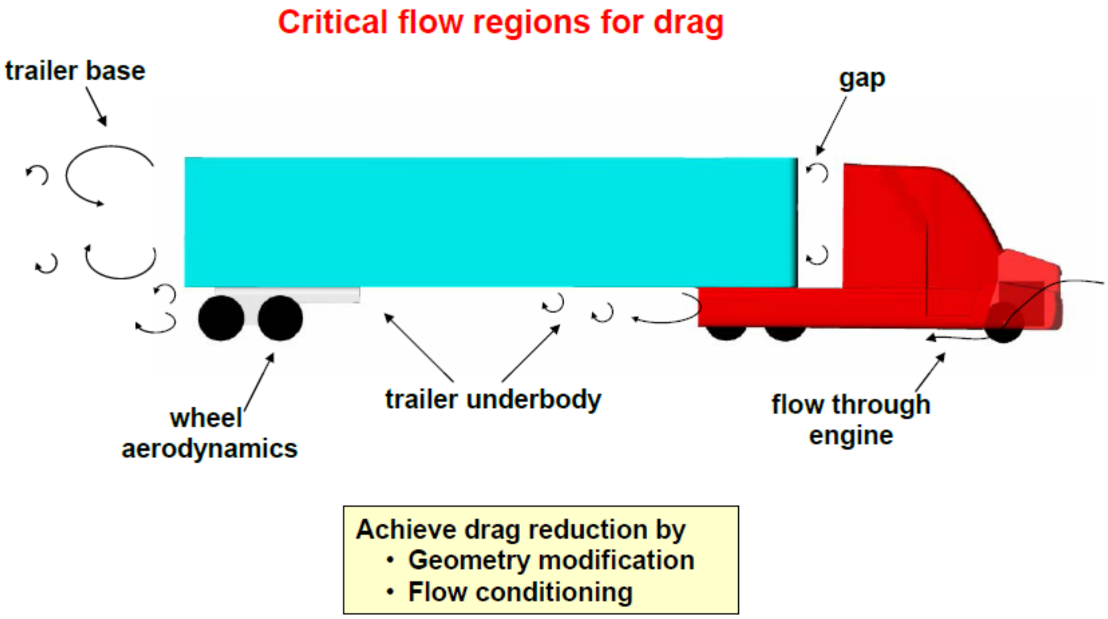


Figure 1.5 Means of achieving drag reduction [1]

1.4.1 Shape Optimization Using a Genetic Algorithm

In this thesis, the first method investigated for truck drag reduction is the shape optimization of the front of the truck. A genetic algorithm, an optimization technique inspired by biological evolution, is employed in conjunction with a computational fluid dynamics (CFD) solver to find the truck front shapes with minimum drag, optimized for certain air speeds and some geometric constraints.

1.4.2 Active Flow Control Using Synthetic Jet Actuators

The second method investigated is active flow control to modify the wake of the truck so as to increase the base pressure and thereby reduce the aerodynamic pressure drag. Three truck-shaped bodies are studied: a two-dimensional generic truck body, a two-dimensional D-shaped bluff body, and the three-dimensional Ahmed body. Synthetic jet actuators (SJAs) are employed for active flow control. These are attached to the back of the truck at appropriate locations. Using a CFD solver, the effect of these active flow control devices is investigated in drag reduction.

Chapter 2

Literature Survey

For modification of fluid flow, both passive and active flow control techniques have been extensively studied for the past seven decades. Passive flow control has been preferred because it does not require external energy input into the fluid system. Passive control devices such as vortex generators, microramps, LEBU, etc., have been studied and have yielded desired control in many applications. However, the success has been limited in achieving significant reduction in drag. Similarly, the success of passive flow control devices such as thrust vectoring and thrust augmentation of ejectors has been limited in reducing flow separation.

Active flow control studies began during World War II. They were mainly applied to military applications of fluid systems. The systems employed reservoirs/sources of fluid from which the fluid was injected into the boundary layer flow, usually over an aircraft wing, or the fluid was sucked in from the boundary layer to the reservoir. Thus, the injection or suction of fluid was employed to alter the boundery layer flow. Because of the complexity of incorporating such an injection/suction system into a wing, these approaches were never implemented on transport aircraft. The key problem was incorporating the source or reservoir for the injection fluid and the tubing/ducting required to inject the fluid into the boundary layer. Even if the engine exhaust could be employed for such a purpose, the complexity of diverting it into the wing for injection into the boundary layer would add considerable complexity and extra weight. Suction of fluid from the boundary layer would also require incorporation of sucking mechanisms and then expulsion of the fluid from the wing again adding complexity and weight.

However, recent active flow control techniques (synthetic jets, pulsed jets, microjets), using periodic excitation technology with zero mass flux but net momentum imparted into the flow, have shown considerable potential for flow modification.

2.1 Large Trucks Drag Reduction Using Active Flow Control

Seifert et al. [5] experimentally studied the active control of flow behind a truck using an “add-on” device at the base of a generic truck body. The cylindrical “add-on” device, shown in Figure 2.1, employs suction and oscillatory blowing to influence and reduce the area of turbulent boundary layer separation behind the truck. The device is a combination of an ejector jet-pump and a bi-stable fluidic amplifier; it was developed and studied by Arwatz et al. [6].

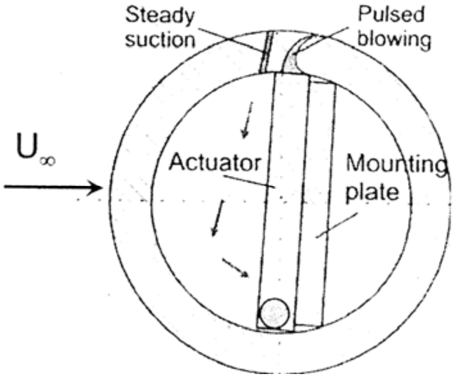


Figure 2.1 Cross section of an active flow control device or actuator employing suction and oscillatory blowing [6]

The experiments were performed on a generic nominal two-dimensional model of a truck in a wind tunnel as shown in Figure 2.2. Two experiments were performed at Reynolds numbers ranging from 200,000 to 1,000,000, corresponding to free stream speeds ranging from 5 to 65 mph. One experiment was performed with the cylindrical active flow control device (shown in Figure 2.1) attached to the top aft corner of the truck and the other experiment was performed with two AFC devices attached at the top and bottom corners on the rear face of the truck. The bottom device employed only steady suction.

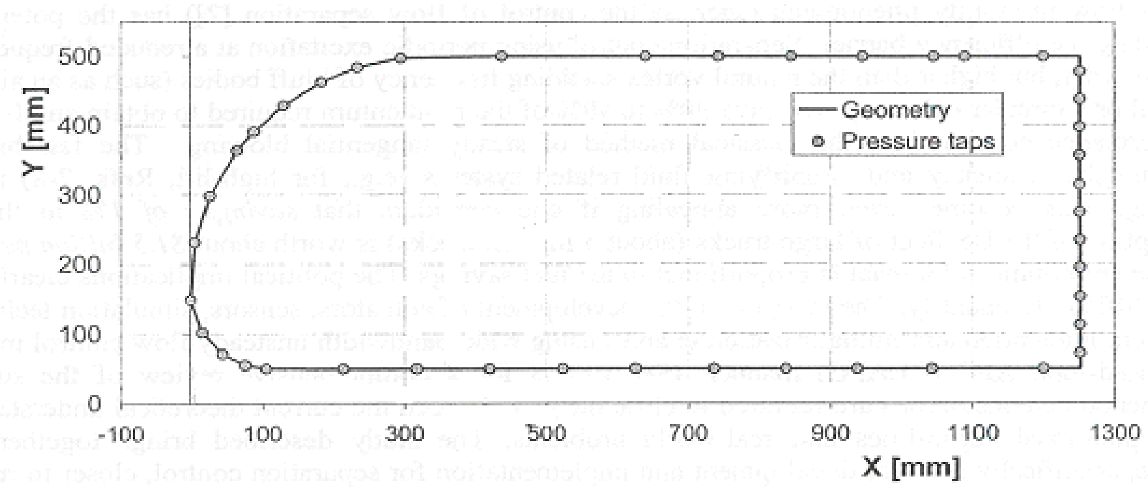


Figure 2.2 Generic truck body [6]

The first experiment showed a 20% aerodynamic drag reduction at lower speeds (*Re* = 200,000), decreasing to 5% reduction at higher speeds (*Re* = 1,000,000). Net power savings increased with Reynolds number while the estimated maximum fuel savings were about 3.7% at 47 mph compared to the case without AFC. The second experiment, with the combination of top and bottom active flow control cylinders, showed 30% drag reduction at lower speeds, decreasing to 20% at higher speeds. This translated to about 10% in net fuel savings.

2.2 Feedback Shear Layer Control for Bluff Body Drag Reduction

Pastoor et al. [7] studied the active flow control on a D-shaped bluff body using feedback shear layer control. The body was mounted in a wind tunnel as shown in Figure 2.3. The top and bottom aft corners were fitted with actuator slots. The slots contained loudspeakers that created a sinusoidal zero-net-mass-flux actuation effect at a 45° angle above and below the direction of the free stream as shown in Figure 2.3.

Using a reduced-order vortex model, Pastoor et al. [7] showed that the initial separation and roll-up of the shear layers in the bluff body wake were dominated by two-dimensional spanwise vortex structures. They found four distinct states in the evolution of the D-body wake. These were startup vortices, shear layer vortices, wake instability, and the vortex street. This final state was consistent with the Kármán vortex street. This yielded a short deadwater region and strongly bent shear layers, giving low pressure in the aft region and consequently high drag.

Pastoor et al. [7] attempted to change the wake formation by forcing synchronous vortex shedding. With periodic active flow control, they achieved a maximum base pressure increase for an actuation Strouhal number (*St*) of 0.15 and momentum coefficient, c_μ, of 0.01 for all Reynolds numbers from 30,000 to 70,000 based on the height of the D-shaped body. This actuation created an elongated deadwater region with symmetrical vortex formation and a delay in the formation of the vortex street. The mean base pressure increased by 40% while the drag decreased by 15%. This resulted in a net power gain of 2.8. In some cases, depending upon the Strouhal number, the base pressure decreased and the drag increased above the baseline values (without AFC).

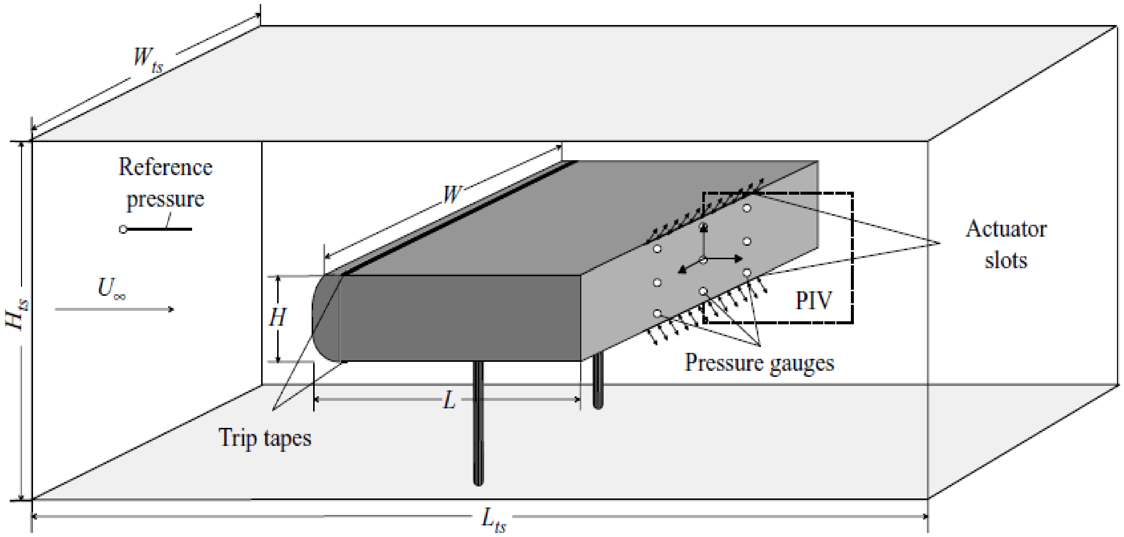


Figure 2.3 D-shaped bluff body [7]

2.3 Application of Slope-Seeking to a Generic Car Model for Active Drag Control

Brunn et al. [8] investigated the reduction of aerodynamic drag on the Ahmed body with active flow control in a wind tunnel. The Ahmed body [9] is a generic car model with a sloped back as shown in Figure 2.4. Its wake depends strongly on the back slope angle. For angles smaller than 30°, the flow first separates from the roof of the body, then reattaches on the slanted surface before separating again at the rear. For angles larger than 30°, the wake is completely separated. As the slant angle approaches 30°, the aerodynamic drag increases sharply. Brunn et al. used a 25° slant angle in their study of the Ahmed body.

The near wake of the Ahmed body is dominated by nearly two-dimensional spanwise vortex structures similar to the generic truck body (section 2.1) and the D-shaped bluff body (section 2.2). The Ahmed body also generates longitudinal vortices that separate from the slant edges. Thus, the wake is highly three-dimensional as shown in Figure 2.4. Brunn et al. [8] used slits with periodic actuators on the middle aft edge at an angle of 45° above the free stream direction and on the lower aft edge at an angle of 45° below the free stream direction. Slits with steady blowing at the slant corners were also employed as shown in Figure 2.5. Each actuator in Figure 2.5 was connected to a closed-loop slope-seeking controller that determined optimum forcing parameters in real-time.

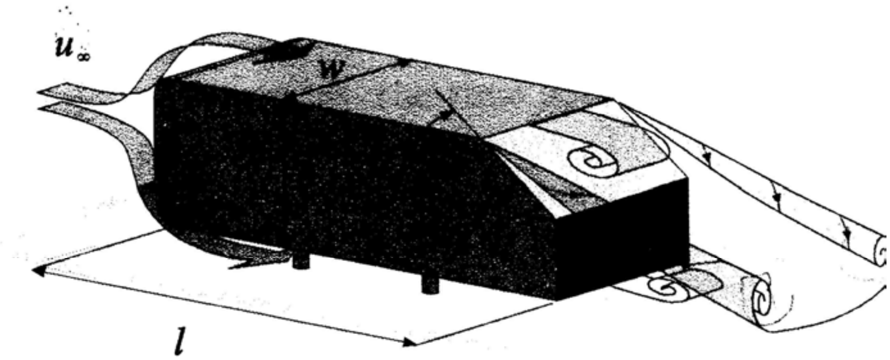


Figure 2.4 Three-dimensional vertical structures behind the Ahmed body [8]

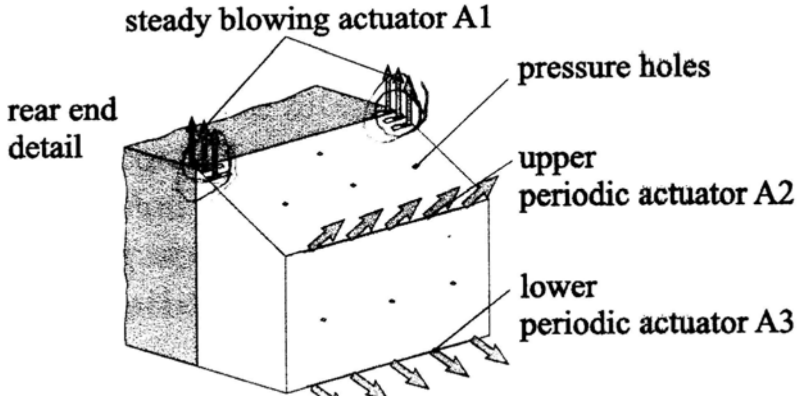

steady blowing actuator A1
pressure holes
rear end detail
upper periodic actuator A2
lower periodic actuator A3

Figure 2.5 Steady and periodic actuators placed on the Ahmed body for active flow control [8]

For a Reynolds number of 400,000, experiments of Brunn et al. [8] found the greatest aft-pressure recovery and drag reduction with the periodic actuators at Strouhal number $St_H = 0.43$ and steady blowing at slant corners at $c_\mu = 0.002$. This corresponded to a pressure recovery of 15% and an overall drag reduction of 13%.

2.4 Drag Reduction of a Generic Car Model Using Steady Blowing

Wassen et al. [10] also studied the active flow control of aerodynamic drag on the Ahmed body by numerical simulation. Using a slant angle of 25° and Large Eddy Simulations (LES), the authors applied steady blowing through slits on all the rear edges of the body as shown in Figure 2.7. At a Reynolds number of 500,000, the authors found a drag reduction of 1.9% on the slant surface and 4.3% on the vertical surface for a total reduction of 6.4% due to steady blowing. The power input required for the steady blowing was 5.6%, thus the net gain in drag reduction was only 0.8%. Losses in pumps or tubing needed for steady blowing would make this reduction even smaller. The authors contend that by applying blowing only at the slant edges, the power input would decrease by 70% and still give a comparable amount of drag reduction.

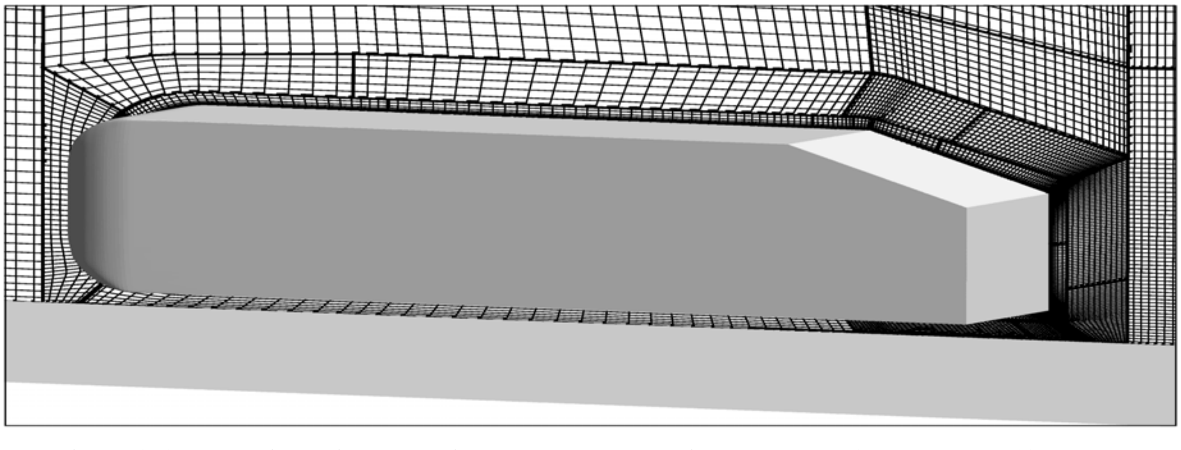


Figure 2.6 Three-dimensional multi-block structured grid around the Ahmed body [10]

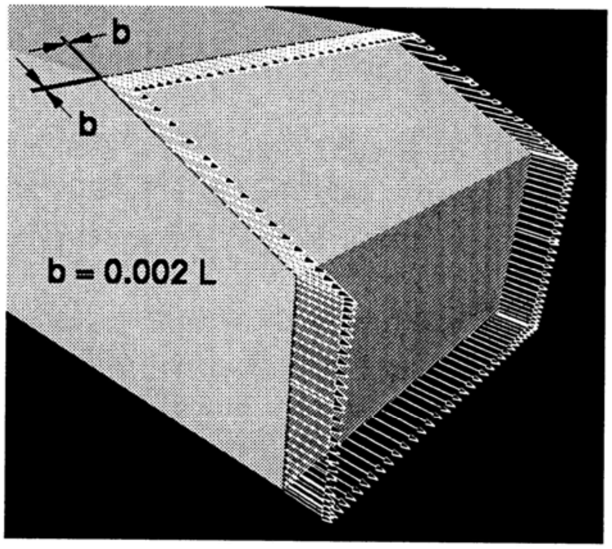


Figure 2.7 Steady blowing actuators on the edges of the slant and vertical surfaces on the backside of the Ahmed body [10]

Chapter 3

Computational Fluid Dynamics (CFD) Solver

Two ANSYS computer programs were used for the numerical simulations. GAMBIT [11] was used to create the truck geometry and mesh while FLUENT [12] was used to solve for the flow field.

3.1 GAMBIT

GAMBIT is the preprocessing grid generation software that is provided with the FLUENT package. It uses a menu- and GUI-based interface. It allows the user to import vertex data as well as CAD models. Sophisticated geometries can be created using its built-in geometry tools.

Meshing is done automatically after the user creates a line mesh for each feature. In two dimensions, GAMBIT has three grid element options that include quadrilateral elements, triangular elements, and a combination of quadrilateral and triangular elements. With these elements, GAMBIT offers five ways to construct meshes. GAMBIT can create a fully structured mesh, an unstructured mesh, or a combination of the two. Each meshing scheme has specific features that make it more suitable for certain types of geometries. Figure 3.1 shows an example of a 2-D regular, structured mesh using only quadrilateral elements. Figure 3.2 shows a 2-D unstructured mesh using only triangular elements and Figure 3.3 shows a 2-D unstructured mesh with quadrilateral elements elements. GAMBIT can also create three-dimensional meshes using hexahedral, wedge, pyramidal, and tetrahedral elements. There are six types of meshing schemes in 3-D in which GAMBIT employs these elements. Similar to two-dimensional meshes, the three-dimensional meshes can be fully structured, fully unstructured, or a hybrid combination using various types of elements.

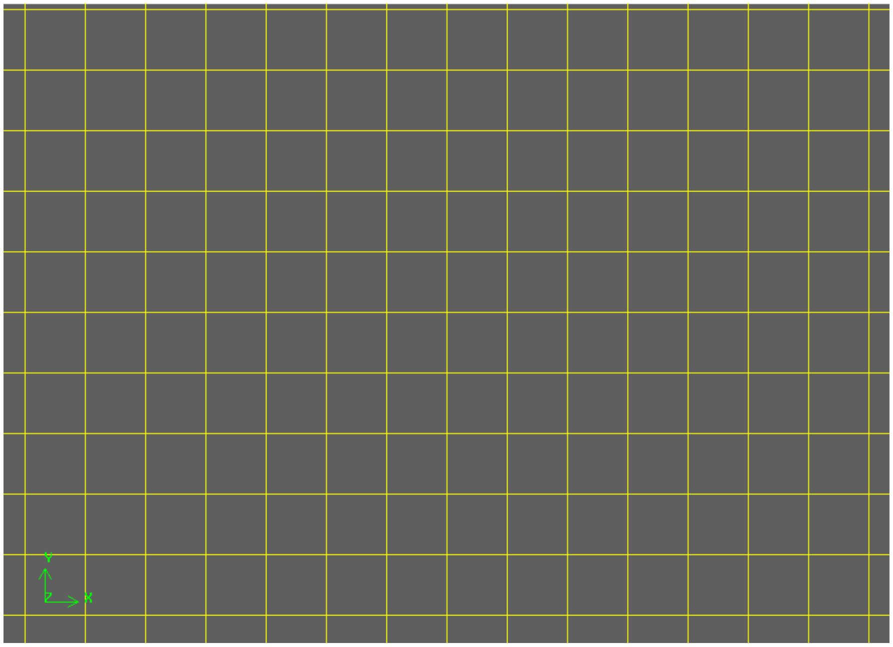


Figure 3.1 Quadrilateral structured grid

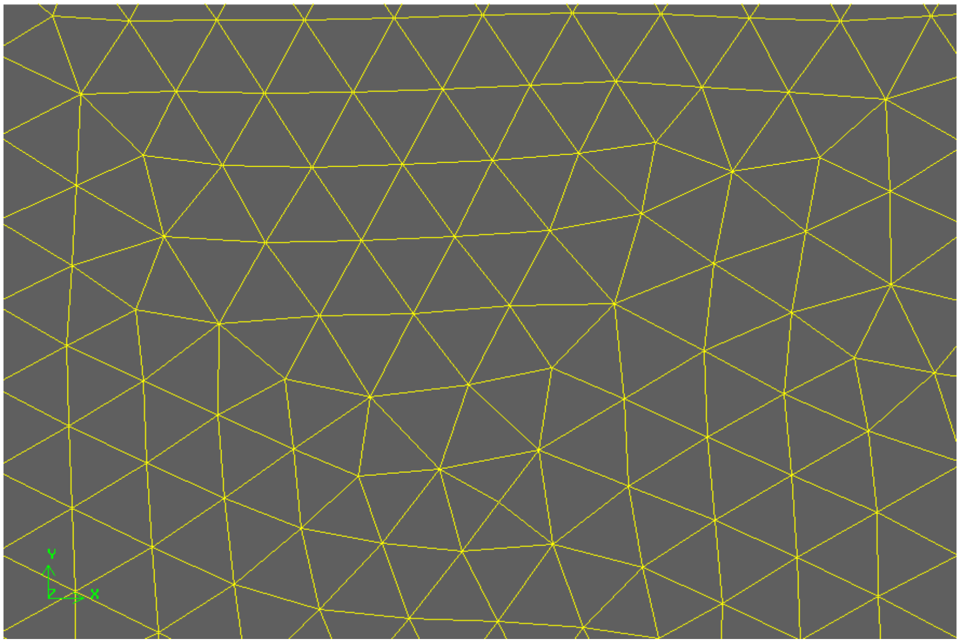


Figure 3.2 Triangular unstructured grid

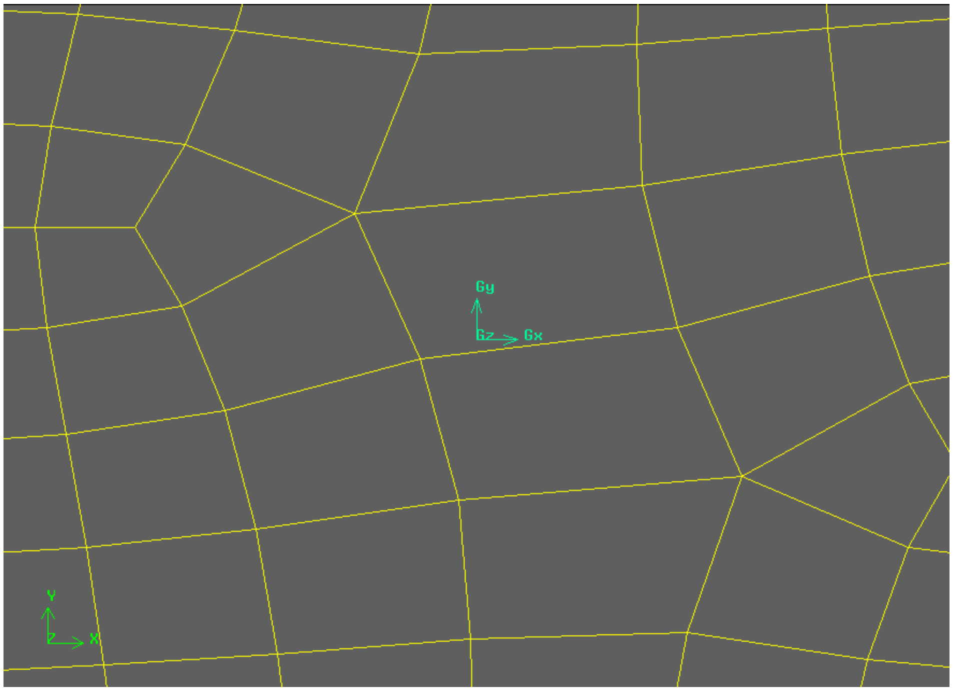


Figure 3.3 Quadrilateral unstructured grid

In addition to meshing the geometry, GAMBIT has tools for grouping the elements of the mesh so that specific boundary conditions can be applied to these groups on various boundaries of the geometry. These boundary conditions include no-slip wall, pressure outlet, velocity inlet, and axis of symmetry conditions among others. All elements that are not assigned boundary conditions are assigned continuum parameters for the medium as fluid or solid. After completion of the meshing and assignment of boundary conditions, GAMBIT allows easy export of the computational model into FLUENT—the CFD solver. All elements and their respective conditions become integrated into a case file for FLUENT processing.

3.2 FLUENT

FLUENT is a computer program that can model fluid flow and heat transfer in complex geometries using an interactive, menu-driven interface. It solves the unsteady Reynolds-averaged Navier-Stokes equations using a finite volume technique. It can solve two- and three-dimensional problems in steady and unsteady simulations. FLUENT has the capability to solve incompressible and compressible flows using inviscid, laminar, and turbulent viscosity models. There are six turbulence models, including the two-equation k-ε and one-equation Spalart-Allmaras models. Each model has options to change parameters as needed for specific cases. FLUENT also allows for adaptation of an existing mesh for additional grid refinement for coupling important features of the flow field with high pressure, density, or velocity gradients without changing the original mesh file from GAMBIT.

FLUENT solves the non-linear fluid dynamics equations by an iterative explicit or implicit algorithm. The converged solution is obtained by specifying a convergence criterion for the residuals of the flow variables such as velocity components, pressure, density, etc. The user can also monitor the output variables such as the coefficient of lift or drag and determine the convergence manually. FLUENT has many post-processing tools that make it easy to visualize and plot the output of simulations. The user can create contour and other plots of flow variables in and around geometries used in the simulation. Pathlines and streamlines can also be computed at any instant of time for unsteady flow simulations. Animations of contour plots and vector plots can also be easily created. The animation tool is very powerful since it translates the simulation data into flow visualization that can be interpreted more easily than columns of output data.

Chapter 4

Passive Flow Control by Shape Optimization Using a Genetic Algorithm

4.1 Introduction

Design optimization is a subject of great interest in the transportation industry since optimization can lead to lighter, faster, and more fuel-efficient transportation vehicles. Due to the highly nonlinear nature of the Navier-Stokes equations, analytical aerodynamic solutions for air or ground transportation vehicle geometry needed for optimizing the objective values such as lift (e.g. aircraft) or drag are not available. Hence, currently almost all practical aerodynamics problems are solved iteratively using specialized CFD software. Furthermore, traditional gradient-based optimization techniques cannot be easily applied; therefore in recent years adjoint methods from control theory [13] and stochastic techniques such as genetic algorithms [14] have been increasingly employed for aerodynamic shape optimization.

This chapter investigates the application of a genetic algorithm to shape optimization of the front part of a generic truck for minimizing its drag coefficient. The space of cubic Bezier curves is employed for defining the front shape of the truck. Initially, the genetic algorithm for shape optimization was developed for maximizing the lift coefficient of a low Reynolds number airfoil employing the design space of Joukowski airfoils. Here, we first describe the shape optimization of low Reynolds number airfoils for maximum lift at a given Mach number and angle of attack for UAV applications. This is then followed by the shape optimization of the front shape of a truck at a given speed for minimizing the drag coefficient.

4.2 Genetic Algorithms

Genetic algorithms are a class of stochastic optimization algorithms inspired by biological evolution. In genetic algorithms, a set or generation of input vectors, called individuals, is iterated over, successively combining traits of the best individuals until a convergence is achieved. In general, GA employs the following steps [13, 15].

1. **Initialization**: Randomly create k individuals. Each individual is a chromosome with n individuals. See Figure 4.1.

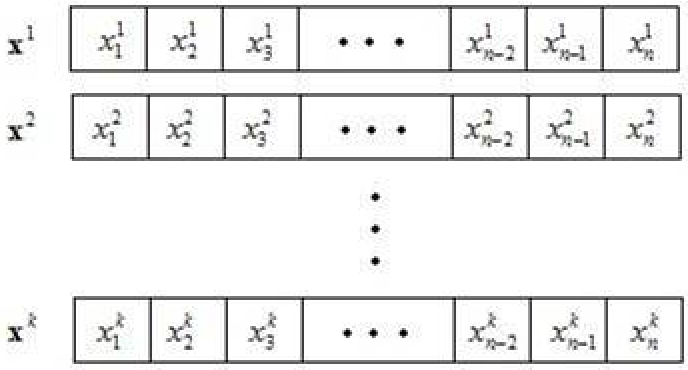

$\mathbf{x}^1$
x_1^1 x_2^1 x_3^1 • • • x_{n-2}^1 x_{n-1}^1 x_n^1
$\mathbf{x}^2$
x_1^2 x_2^2 x_3^2 • • • x_{n-2}^2 x_{n-1}^2 x_n^2
$\mathbf{x}^k$
x_1^k x_2^k x_3^k • • • x_{n-2}^k x_{n-1}^k x_n^k

Figure 4.1 Individuals with n alleles

2. **Evaluation**: Evaluate the fitness of each individual.
3. **Natural Selection**: Sort the individuals from best to worst fitness. Remove a subset of the individuals. Often the individuals that have the worst fitness are removed. However, culling, the removal of individuals with similar fitness is sometimes performed.
4. **Reproduction**: Pick pairs of individuals to produce an offspring. This is often done by roulette wheel sampling; that is, the probability of selecting some individual h_i for reproduction is given by:

$$P[h_i] = \frac{fitness(h_i)}{\sum_i fitness(h_i)}. \tag{3}$$

A *crossover* function is then performed to produce the offspring. Generally, the crossover is implemented by choosing a crossover point on each individual and swapping alleles – or vector elements – at this point as illustrated in Figure 4.2.

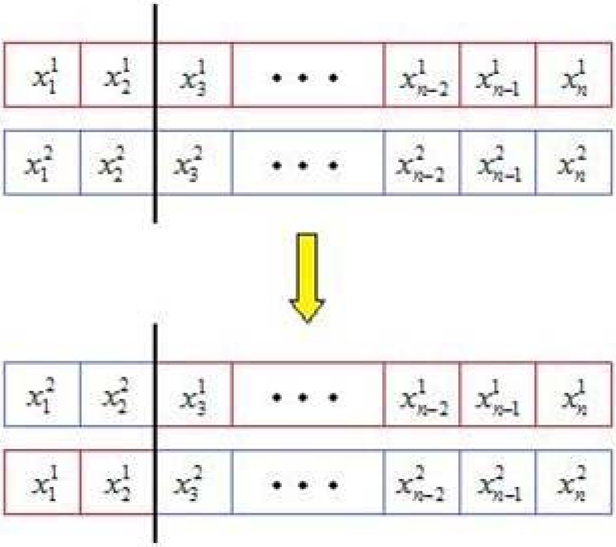


Figure 4.2 General Crossover

5. **Mutation**: Randomly alter some small percentage of the population as shown in Figure 4.3.

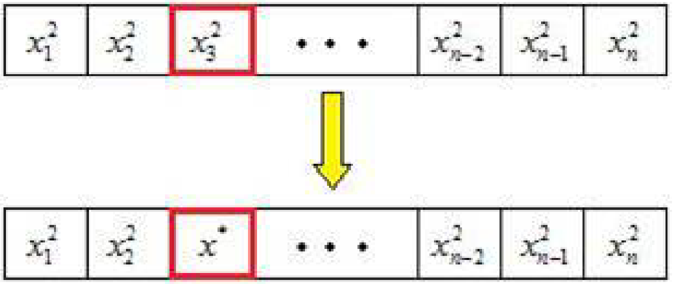


Figure 4.3 Mutation

6. **Check for Convergence**: If the solution has converged, the best individual observed is returned. If the solution has not yet converged, label the new

generation as the current generation and go back to step 2. Convergence is often defined by computation of a certain number of generations or a similarity threshold.

4.3 Airfoil and Truck Parameterization

4.3.1 Joukowski Transformation for Airfoil Parameterization

The Joukowski transformation [16] provides an easy mechanism for describing an airfoil by a small number of geometric parameters. It allows the definition of an airfoil in the standard coordinate system by transforming a circle in a conformal plane. The conformal coordinate system is denoted as the ζ-plane with ξ and η axes. The transformation then maps a circle centered in the second quadrant that intersects the positive ξ-axis at point (c, 0) to an airfoil in the z-plane by the following functions:

$$x = \xi\left(1 + \frac{c^2}{\xi^2 + \eta^2}\right) \tag{4}$$

$$y = \eta\left(1 - \frac{c^2}{\xi^2 + \eta^2}\right) \tag{5}$$

This transformation has several desired properties. If the center of the circle is in the second quadrant and the circle intersects the positive ξ-axis, then the airfoil will have a sharp trailing edge. The transformation also ensures that the camber line of the airfoil will start and end on the x-axis, removing the need for any further translations or transformations. Figure 4.4 illustrates the application of the Joukowski Transformation.

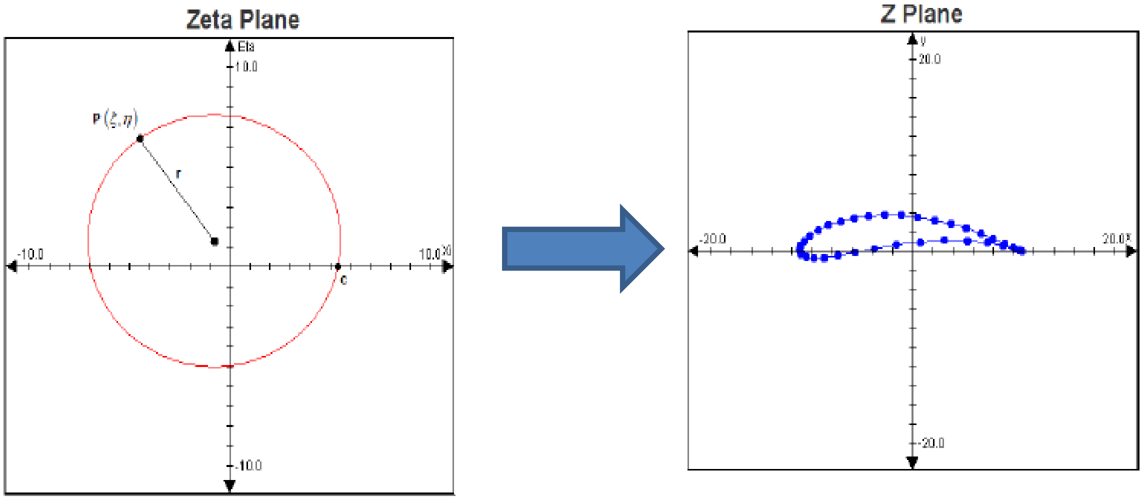


Figure 4.4 Joukowski Transformation

4.3.2 Bézier Curves for Truck Front Parameterization

The Joukowski transformation illustrates that a small set of control points can describe a shape made of many points. Each control point has a large influence on the shape produced. Thus, the genetic algorithm can find the optimal shape quickly by only having to deal with a small number of points. This idea can be applied to other optimization problems.

The Bézier curve is a parametric curve discovered in 1962 by the French engineer Pierre Bézier. He originally used the curves to design automobiles. The Bézier curve utilizes two anchor points and *n* control points. The anchor points, P_0 and P_n, determine where the curve begins and ends. The control points, $P_1 \ldots P_{n-1}$, influence the curve's path between the anchor points. The transformation from anchor and control points to a line is given by:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^{i} P_i, \qquad t \in [0,1]. \tag{6}$$

Just as Bézier used the above parameterization to design automobiles, this parameterization has been employed in this work to describe the shape of a truck front. The parameterization uses only two control points, P_1 and P_2, as shown in Figure 4.5

and is thus a cubic Bézier curve. The anchor points are kept at constant *x* and *y* values while the control points are generated and improved by the genetic algorithm.

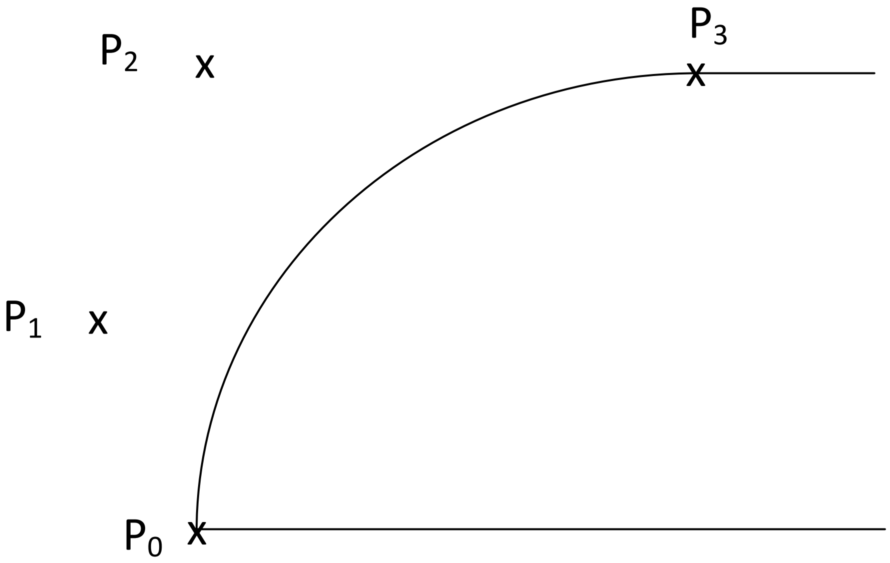


Figure 4.5 Bézier curve cubic parameterization

4.4 Algorithm Implementation

In this section the computational setup is described. Figure 4.6 schematically illustrates how the genetic algorithm (GA) interfaces with the external mesh generation and CFD flow solver.

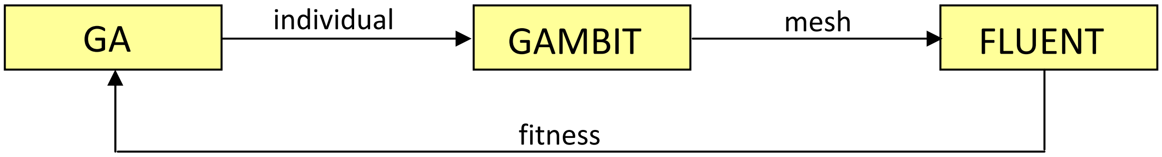


Figure 4.6 Illustration of information flow in genetic algorithm implementation process

A genetic algorithm individual is represented by either an airfoil or truck geometry data file. The file is passed to GAMBIT, which is employed to create a two-dimensional structured, unstructured, or hybrid mesh. The GAMBIT mesh is then transported as input to FLUENT for computation of the flow field. FLUENT is employed iteratively to solve for the coefficient of lift C_l or the coefficient of drag C_d. The solution process, shown in Figure 4.6, is continued until the convergence in the desired objective value is achieved.

In our calculations, a generation size of 10 individuals was used. Each generation had a mutation rate of 5% and a natural selection rate of 50% with no culling tolerance; that is, there was no attempt made to remove similar airfoils or trucks. Since the objective value could take both positive and negative values, roulette wheel sampling could not easily be used for selecting reproducing individuals. Instead, the lowest 50% of each generation were removed and reproduction was done by randomly selecting two individuals. Note, however, that the fittest individuals were still most likely to reproduce because the top 50% of each generation perpetuated to the next generation. The offspring individual was then obtained by stepping a random amount in the direction of the fitter parent according to equation (10) below and shown in Figure 4.7.

$$crossover(\mathbf{x}_1, \mathbf{x}_2) = rand(0,1) \cdot (\mathbf{x}_1 - \mathbf{x}_2) + \mathbf{x}_2 \tag{10}$$

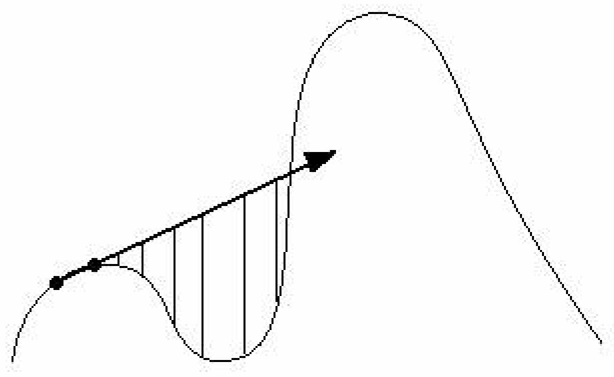


Figure 4.7 Extrapolation-based crossover

Convergence was determined when the fitness of all individuals in a generation differed by no more than 0.001, which usually occured between 150 and 250 generations, or when 250 generations had passed. This second constraint on convergence was applied to prevent the algorithm from running for unreasonably large amounts of time.

4.4.1 Airfoil Constraints

The Joukowski transformation was used to parameterize an airfoil with three parameters: ξ, η, and r. All airfoils were scaled to have a chord length of unity with constraints on the parameters described by equations (7-9). Equation (7) constrained the airfoil search space by preventing the creation of infinitely thin airfoils. Equation (8) constrained the search space to the upper half plane so that the negatively cambered airfoils were not considered. Equation (9) eliminated "fat" airfoils from the search space by limiting the thickness to chord ratio to 20%.

$$-10 \le \xi \le -0.35\,, \tag{7}$$

$$0 \le \eta \le 10\,, \tag{8}$$

$$\frac{t}{c} = -\frac{\sqrt{\eta^2 + \xi^2}}{0.77\left(\sqrt{r^2 - \eta^2} + \xi\right)} \cos\left(\frac{\pi}{2} + \tan^{-1}\left(-\frac{\xi}{\eta}\right)\right) \le 0.20\,, \tag{9}$$

4.4.2 Truck Constraints

The Bézier curve was used to parameterize a truck front with four coordinates of the parameters: $P_1(x_1,y_1)$ and $P_2(x_2,y_2)$. The anchor points, P_0 and P_3, were held fixed at specific points during the optimization. Their *y*-positions relative to each other were held at -0.222 and 0.222, respectively. P_3 was held at -0.954 along the *x*-direction while P_1 changed positions along the *x*-direction for different cases considered. P_1 and P_2 *y*-coordinates were constrained to be random within the boundaries of the P_0 and P_3 *y*-coordinates. P_1 and P_2 *x*-coordinates were held on the right of the P_3 *x*-coordinate and on the left by boundaries set differently for each optimization.

4.5 Mesh Generation

GAMBIT was used to generate meshes with combined structured and unstructured grids for each airfoil and truck considered by the genetic algorithm. A journal file was used to automatically produce a mesh that FLUENT could use to evaluate an airfoil or truck. The journal had to be robust enough to create a usable mesh for any airfoil or truck front shape lest the algorithm would stop running. It took many tries to create a generic mesh journal file that would work for any shape. For both the airfoil and truck fronts, completely structured meshes were tried, but were found to only work for certain families of shapes. An unstructured mesh had to be used to conform to any type of shape.

Thus, the final mesh journal file used a combination of structured and unstructured grids. The faces of the grid not attached to the airfoil or truck front were meshed using the structured quadrilateral cells. It was required that the number of nodes on opposite faces were identical. To ensure this distribution, a set number of nodes were defined (not relative node spacing) along each edge. The faces of the grid attached to the airfoil or truck were meshed using the unstructured triangular cells. This allowed any airfoil or truck front shape that was produced to be meshed without errors that would stop FLUENT from running.

The full mesh for the randomly created airfoil is shown in Figure 4.8. The airfoil is placed in the middle of the square with respect to the y-axis. Its distance from the left and right far-field boundaries is 12.5c and 20c respectively, where c is the chord length of the airfoil. The airfoil sits on a face that has an unstructured triangular mesh as shown in Figure 4.9. Rectangular faces surround the airfoil face. Each rectangular face uses structured quadrilateral cells. Each cell in contact with the airfoil is split into four equal-sized smaller cells as shown in Figure 4.10. This cell adaptation increases the accuracy of the solver in the critical boundary layer region where viscous effects are dominant. The resulting mesh had a node/cell/face count of 15,990/30,485/14,495.

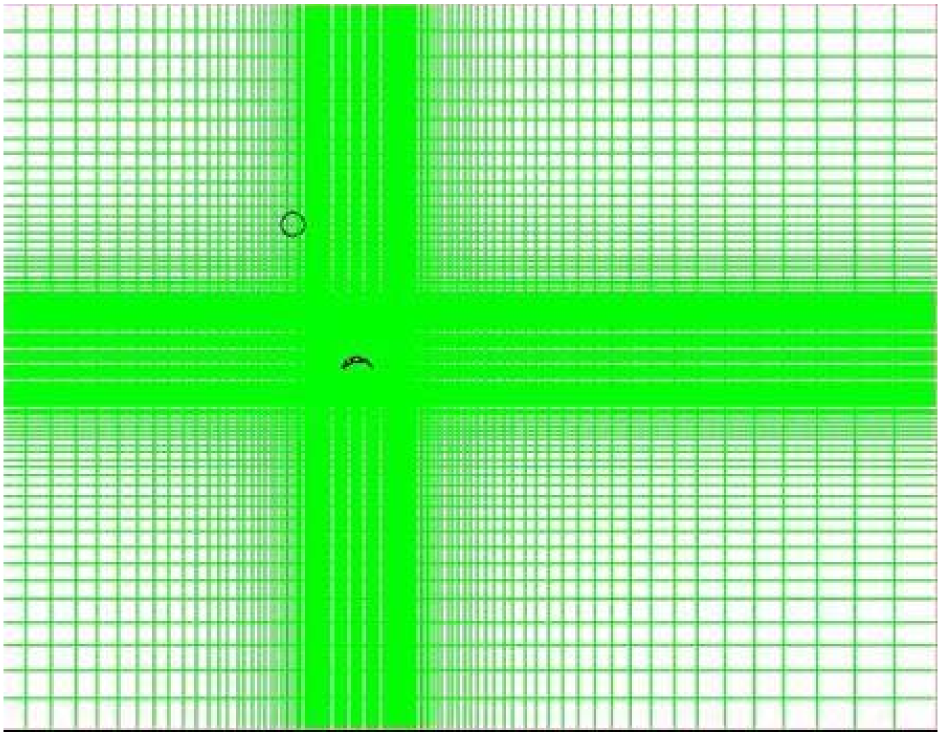


Figure 4.8 Automatically generated adaptive mesh for airfoil

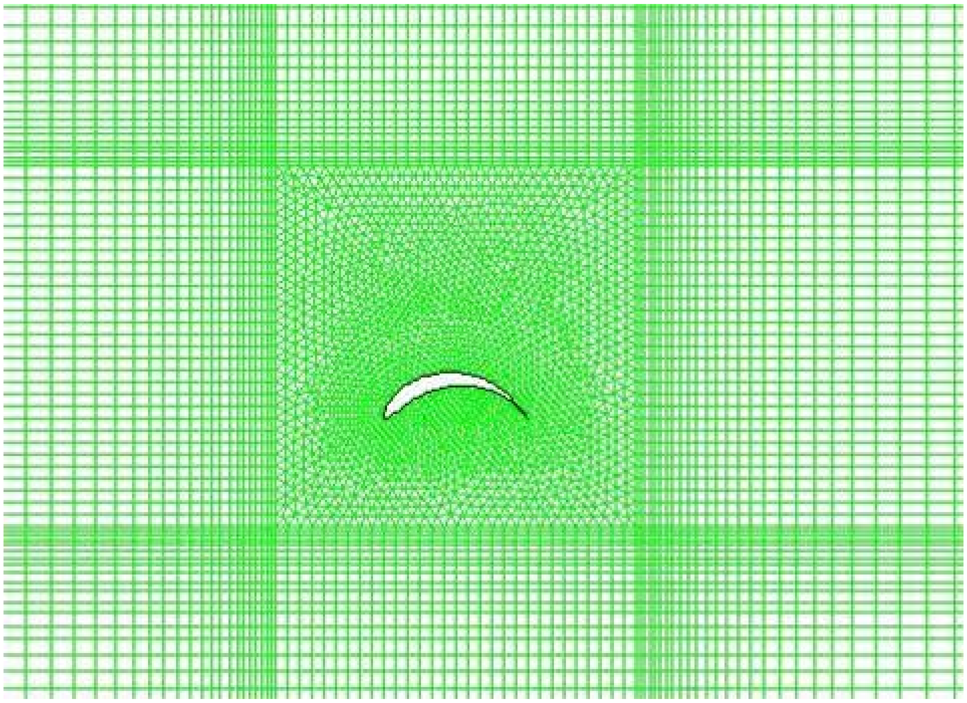


Figure 4.9 Automatically generated adaptive mesh for airfoil, zoomed-in view

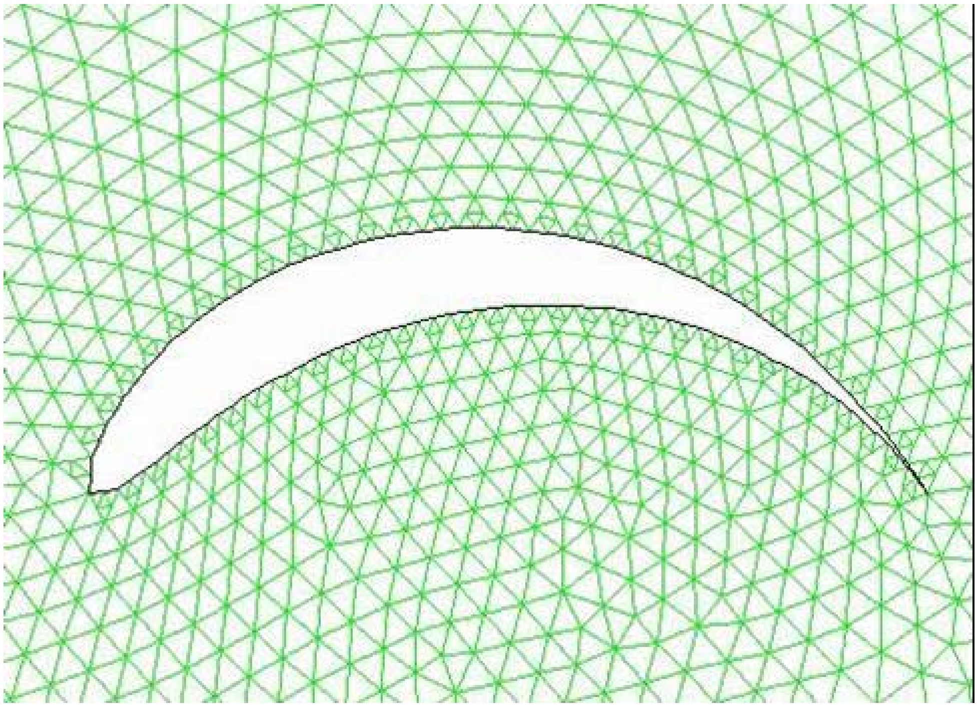


Figure 4.10 Automatically generated adaptive mesh for airfoil, zoomed-in view

The full mesh for the truck is shown in Figure 4.11. The truck is placed at the center of the semi-circle with radius 12.5L, where L is the length of the truck base. Rectangular faces border the semi-circle and have length 20L. The truck front, which is randomly created in the algorithm, is bordered by a face with an unstructured triangular grid as shown in Figure 4.13. The cells immediately surrounding the truck are split into four equal-sized cells as shown in Figure 4.12. This cell adaptation increases the accuracy of the solver in the critical areas where drag-inducing flow structures are dominant. It also increases the accuracy at the back of the truck where bluff body wake features are dominant. The resulting grid had a node/cell/face count of 38,176/68,330/30,154.

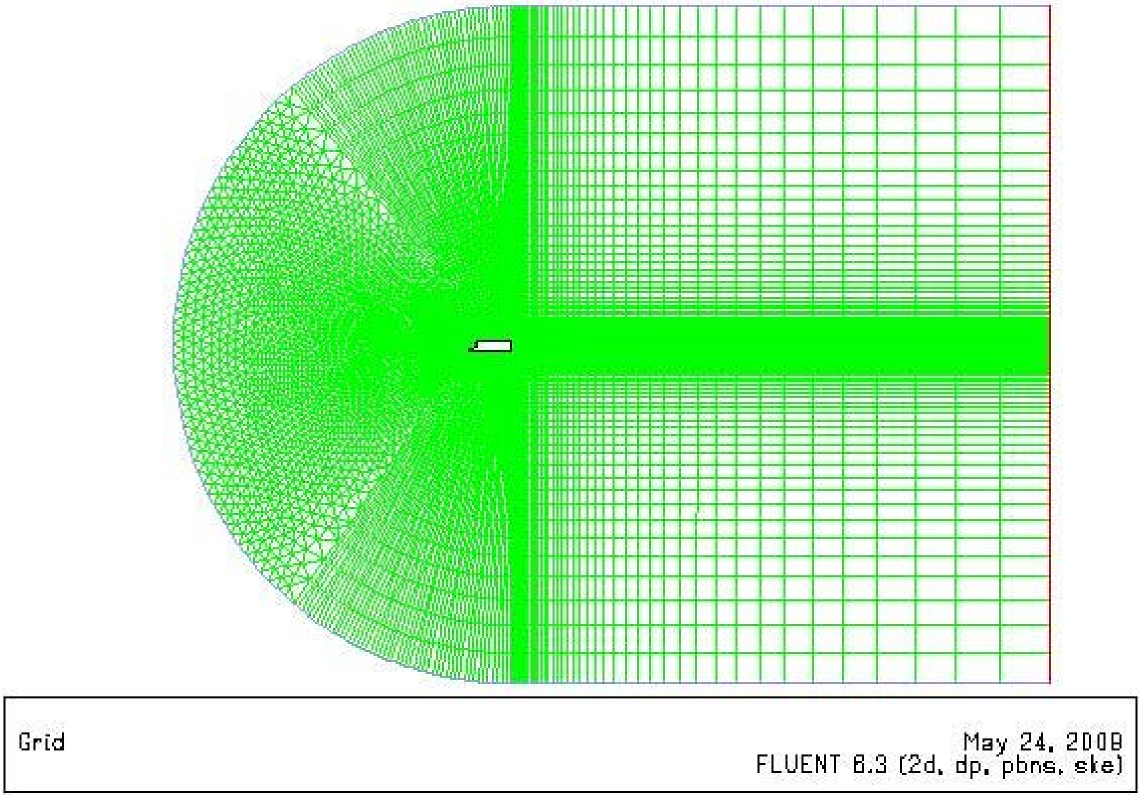


Figure 4.11 Automatically generated adaptive mesh for truck

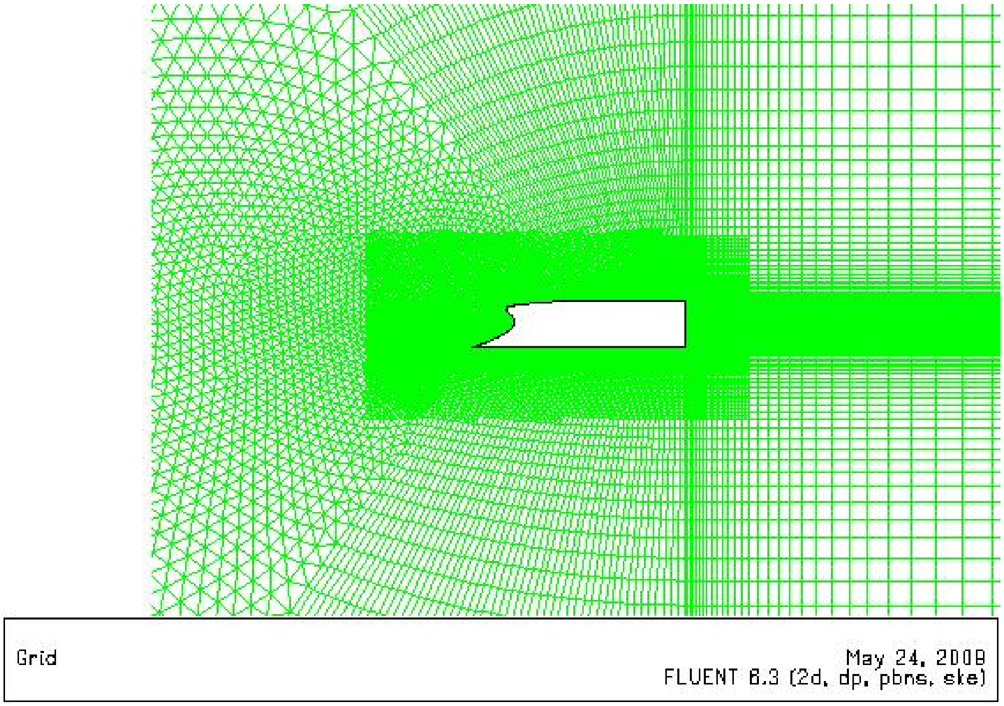


Figure 4.12 Automatically generated adaptive mesh for truck, zoomed-in view

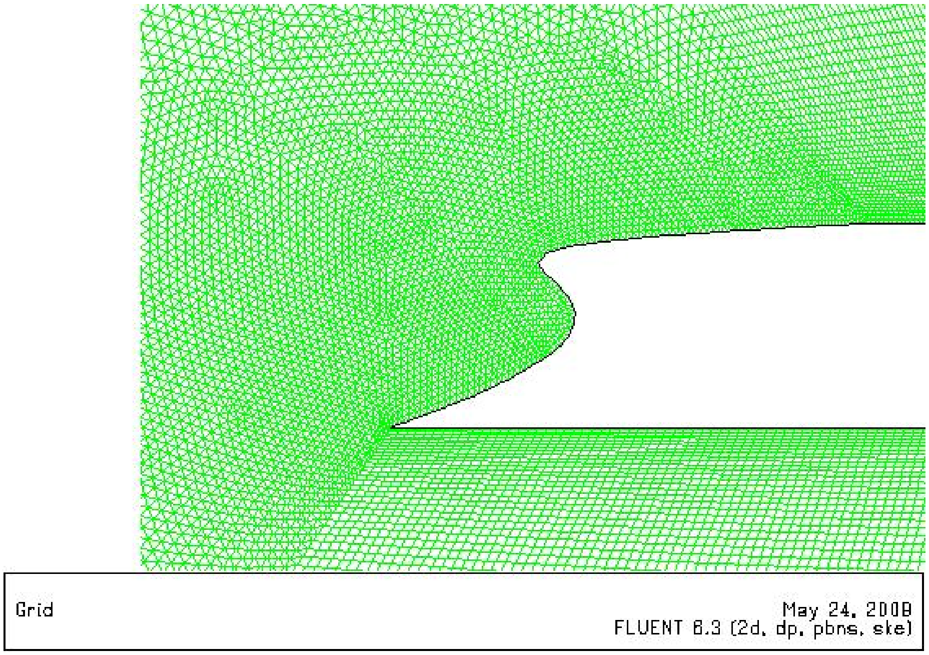


Figure 4.13 Automatically generated adaptive mesh for truck, zoomed-in front view

4.6 Flow Field Computation

FLUENT was used to solve for the coefficients of lift and drag for a given airfoil, and for the coefficient of drag for a given truck. A journal file was used to automatically initialize and evaluate each case while saving a record of the relevant coefficients, C_l and C_d. For the airfoil cases, the FLUENT journal initialized the calculations for a Reynolds number of 100,000 with the chord as the characteristic length. The truck cases used a Reynolds number of 303,957 with the maximum truck height as the characteristic length. The airfoil had a chord length of 10 cm while the truck had a height of 0.45 m. Temperature and static pressure for the truck cases were defined to be at sea level conditions with an ambient temperature of 298.5 K and an ambient pressure of 101,325 Pa respectively. The airfoil cases used the same conditions; this would be the case for a vehicle whose maximum altitude would not exceed 100 meters. At this temperature and pressure, density of air $\varrho = 1.184$ kg/m^3 and the dynamic viscosity of air $\mu = 1.849\text{x}10^{-5}$ kg/m-s were used. For the airfoil, the flow was assumed to be laminar because of its low Reynolds numbers. The laminar viscosity was calculated from Sutherland's law:

$$\mu = \mu_0 \left(\frac{T}{T_0} \right)^{3/2} \left(\frac{T_0 + S}{T + S} \right), \tag{10}$$

where μ_o = 1.716x10^{-5} kg/m-s, T_o = 273 K, and S =111 K. For the truck, the flow was assumed to be turbulent. While the flow at the front of the truck was laminar, the boundary layer separation and consequent roll up of vortices behind the truck caused the wake flow to be turbulent. To model the turbulent flow, the k-ε turbulence model was employed. The turbulent eddy viscosity was calculated with:

$$\mu_t = \rho C_\mu \frac{k^2}{\varepsilon}, \tag{11}$$

where C_μ = 0.09.

4.7 Results and Discussion

4.7.1 Airfoil Optimization

The primary objective of this study was to generate optimized low Reynolds number airfoils that maximized the coefficient of lift at 0° and 2° angles-of-attack. At first, a first-order accurate Navier-Stokes solver and a standard pressure solver FLUENT were used. Later, to investigate the solution dependence on mesh size and the order-of-accuracy of the solver, the mesh density was doubled and a first-order Navier-Stokes solver was employed along with an improved pressure solver, PRESTO, in FLUENT. Table 4.1 summarizes the results.

Table 4.1 Results for C_l - optimized airfoils, Re_c = 100,000

Angle of attack, *a* (°)	Optimized C_l	*t/c* (%)
0	2.35	11.5
2	3.05	10.2

Angle of attack, 0° The first airfoil case optimized the airfoils at a Reynolds number of 100,000 and angle of attack of 0°. The coefficient of lift was found to be 2.35 and the thickness to chord ratio was 11.5 percent. Figure 4.14 shows that the airfoil is moderately cambered, increasing the effective angle of attack and consequently, the coefficient of lift. While increasing camber can cause separation of flow near the trailing edge, this optimized airfoil keeps the flow attached, eliminating the added drag due to separation. Figure 4.15 shows the pressure distribution on the C_l-optimized airfoil. Figure 4.16 shows the convergence plot of the C_l-optimized airfoil; the coefficient of lift eventually attains the maximum value as the number of generations increases. The genetic algorithm does a good job of quickly finding the range of airfoils in which the optimized airfoil will occur. After two generations the genetic algorithm finds the optimal camber for the airfoil. It then increases the thickness of the airfoil for increased coefficient of lift, but does not create shapes that allow the flow to separate.

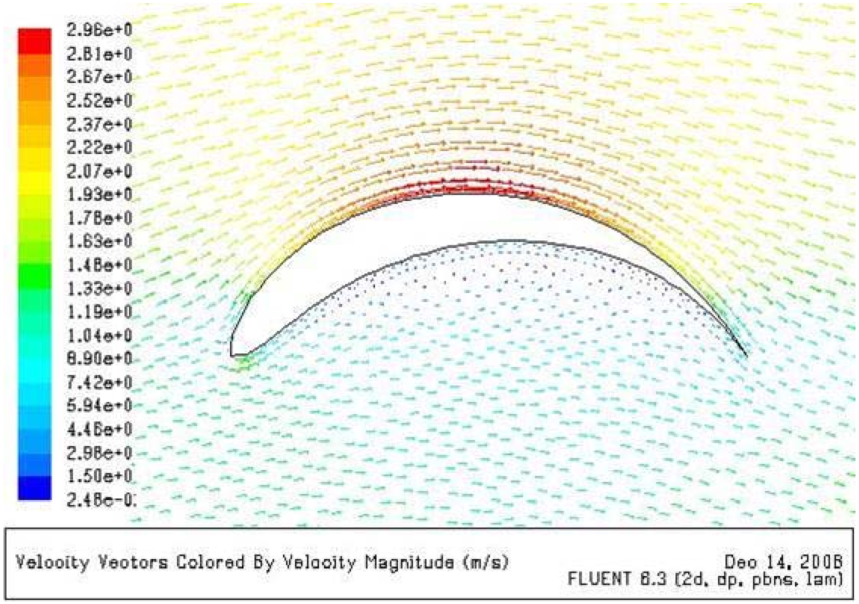


Figure 4.14 Velocity vectors for the C_l-optimized airfoil at $\alpha = 0°$, $Re_c = 100,000$

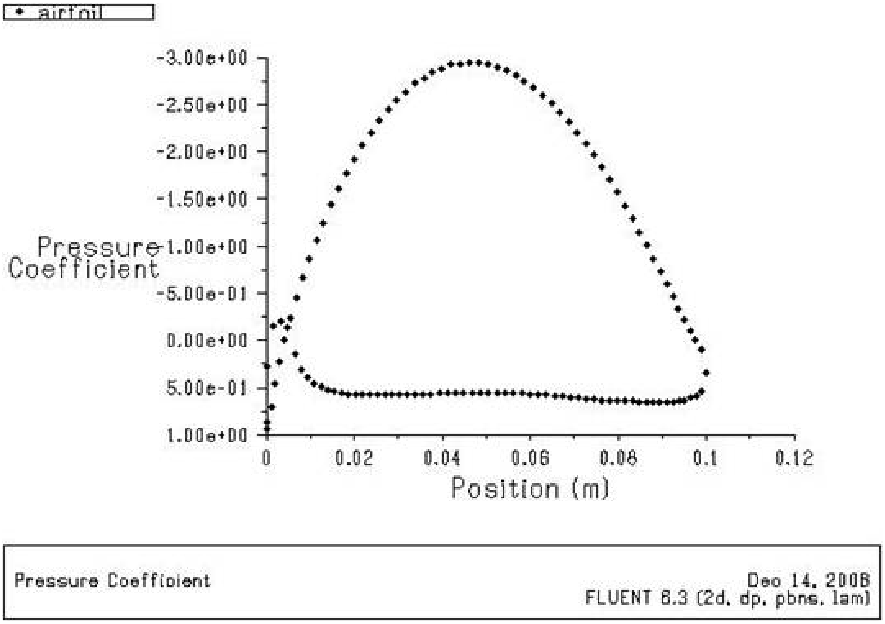


Figure 4.15 Pressure coefficient plot for the C_l-optimized airfoil at α = 0°, Re_c = 100,000

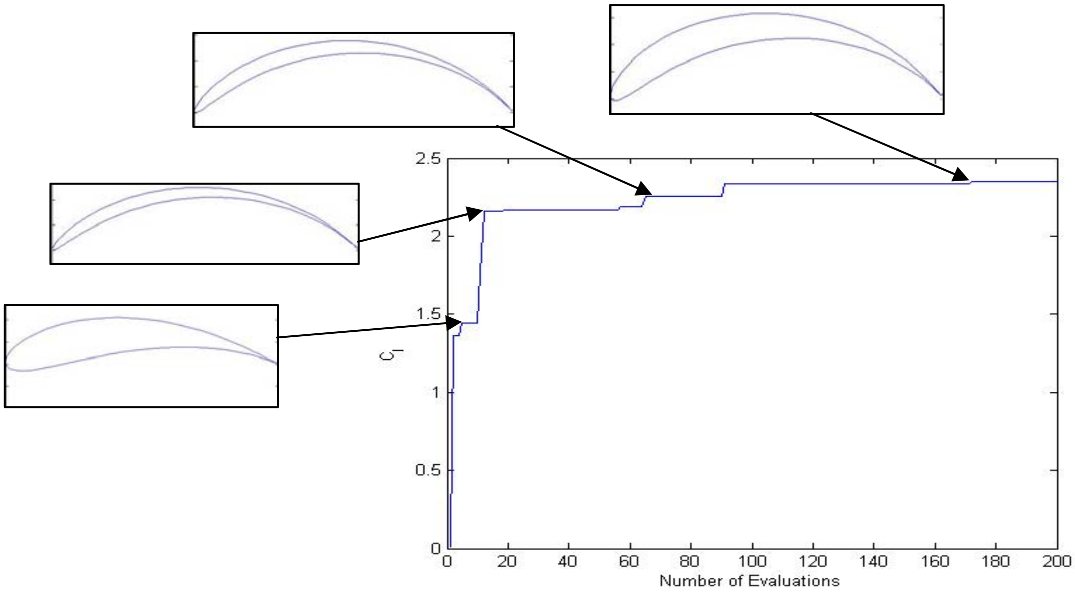


Figure 4.16 Convergence plot for the C_l-optimized airfoil at α = 0°, Re_c = 100,000

Angle of attack, 2° The second airfoil case optimized the airfoils at a Reynolds number of 100,000 and an angle of attack of 2 degrees. The optimized coefficient of lift was found to be 3.05 with a thickness to chord ratio of 10.2 percent. This airfoil looks similar to the airfoil optimized at 0° angle of attack except that it has a smaller thickness. This is because the airfoil experiences more drag at the higher angle of attack, so thickness decreases to lower that drag. The increased angle of attack naturally increases the coefficient of lift. Figure 4.17 shows the shape of the C_l-optimized airfoil with velocity vectors. Figure 4.18 shows the pressure distribution on the optimized airfoil and Figure 4.19 shows a convergence plot showing the evolution of the airfoils as the number of generations increases.

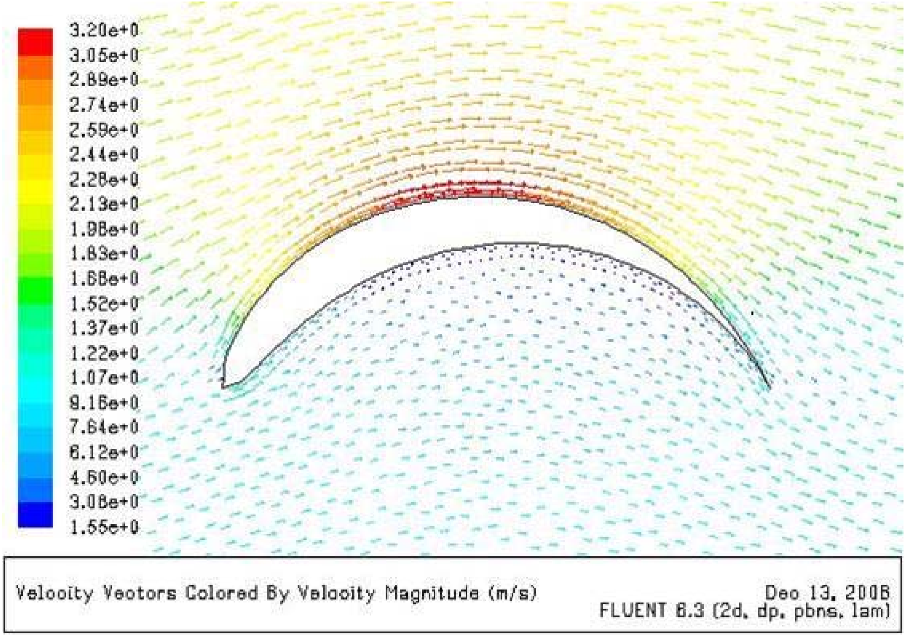


Figure 4.17 Velocity vectors for the C_l-optimized airfoil at $\alpha = 2°$, $Re_c = 100,000$

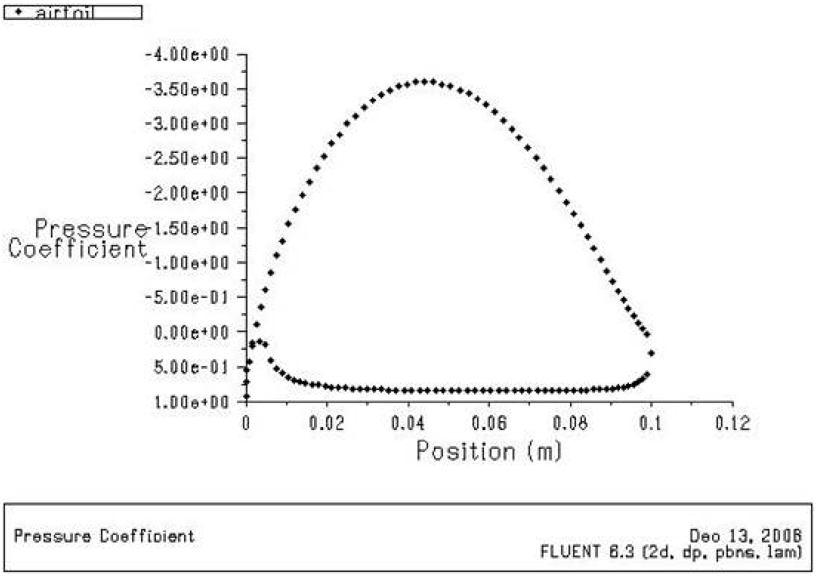


Figure 4.18 Pressure coefficient plot for the C_l-optimized airfoil at $\alpha = 2°$, $Re_c = 100{,}000$

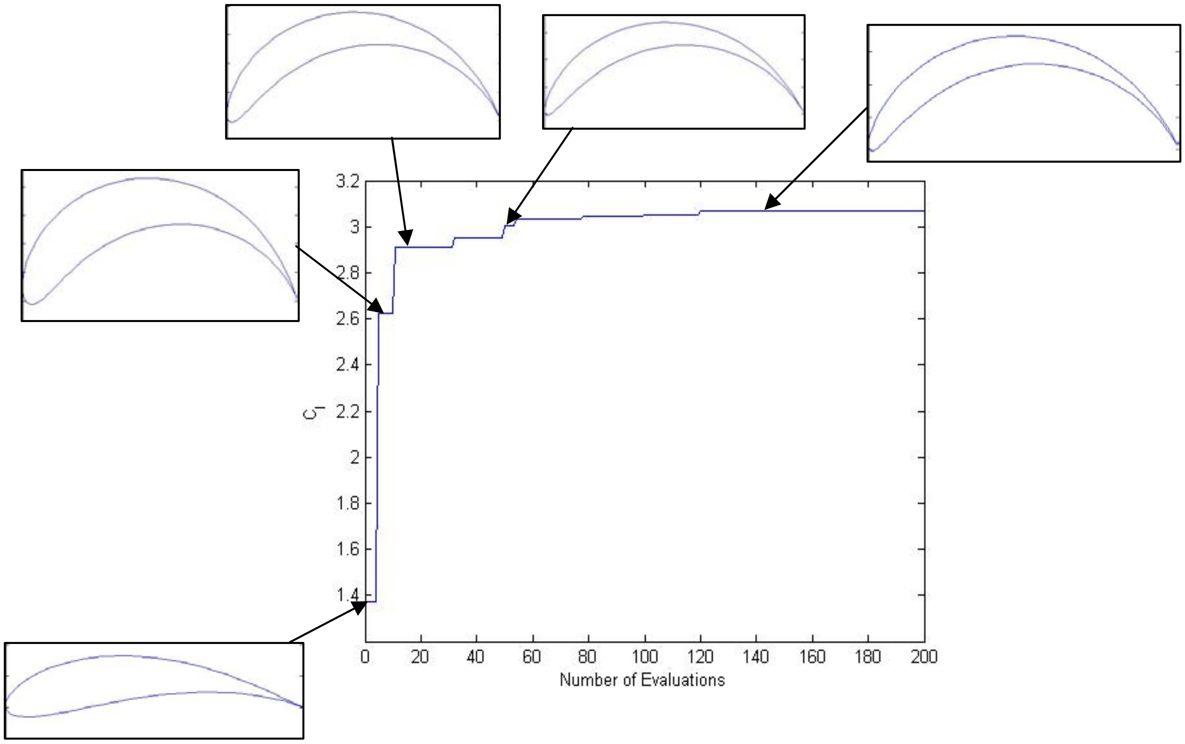


Figure 4.19 Convergence plot for the C_l-optimized airfoil at $\alpha = 2°$, $Re_c = 100{,}000$

4.7.2 Truck Front-Shape Optimization

Five cases were run to find optimal front shapes of the truck. Each case had different x_0 and *xMin* constraints. The five cases are shown in Table 4.2. The baseline case for this study was the generic truck of section 2.1. Its coefficient of drag was 1.045; its calculation is discussed later in section 5.4.

Table 4.2 Results for C_d-optimized truck front

Case	x_0	*xMin*	Optimized C_d	% Change in C_d compared to baseline
1	-1.145	-2	0.789	-24.5
2	-1.5	-2	1.009	-3.4
3	-1.5	-3	1.033	-1.1
4	-2	-3	1.056	+1.1
5	-2	-4	0.712	-31.9

Case 1 It optimizes the truck front shape with a separation of 0.191 between P_0 and P_3 in the *x*-coordinate. The left boundary for the control points was kept at $x = -2$. Figure 4.20 shows the convergence plot showing the evolution of generations towards the optimized shape. The genetic algorithm finds the range of optimal truck fronts within the first generation and then tweaks the shapes to find the best truck front shape within that range. Figure 4.21 shows the velocity contours surrounding the optimized truck front. The shape allows air to flow smoothly over the top edge with no separation at P_3. The bottom edge slightly obstructs the flow creating a low velocity bubble near the nose and some separation at P_0. The coefficient of drag was calculated to be 0.789, which is a 24.5% drop from the baseline case.

While this seems like a great improvement over the baseline case, there are hidden fitness values which the genetic algorithm does not address. For example, where would the engine go in this truck? Where would the driver sit? It would not be easy to convince the trucking industry that they should drastically change their truck cab designs. It would be easier to find a truck front shape that improves on exisiting designs, but still has the same basic shape as trucks that are in service today. Thus, although this truck front shape shows marked improvements over the baseline truck, it is not an ideal improvement.

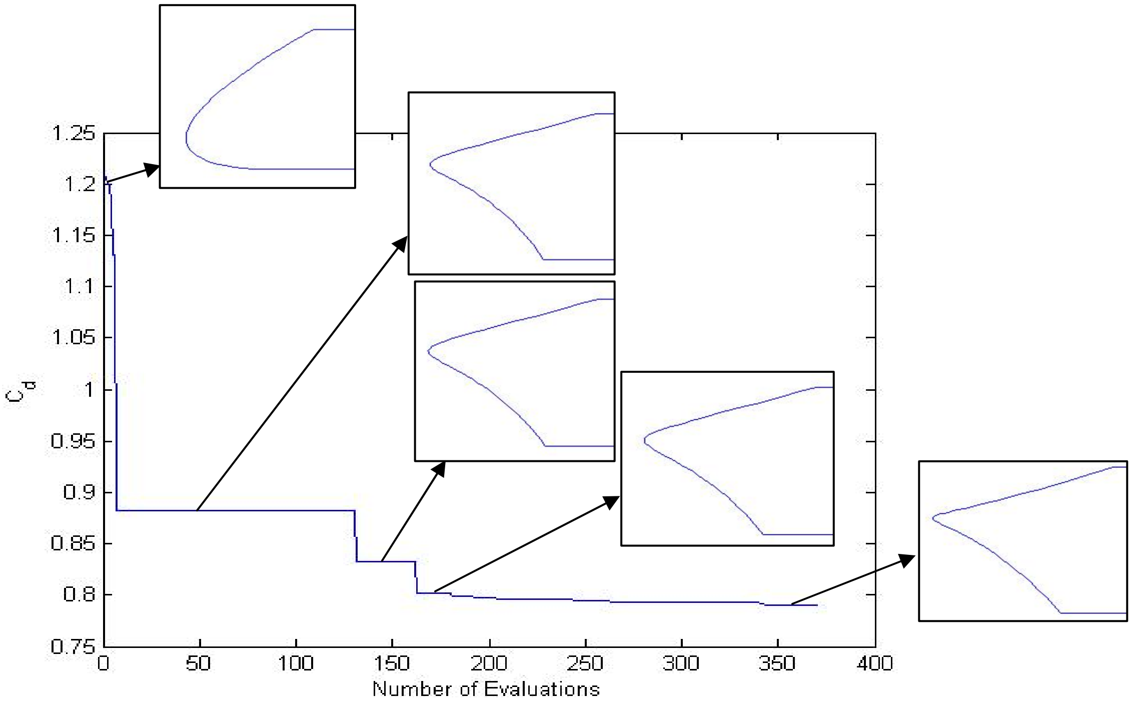


Figure 4.20 Convergence plot for Case 1 C_d-optimized truck front, Re_H = 303,957

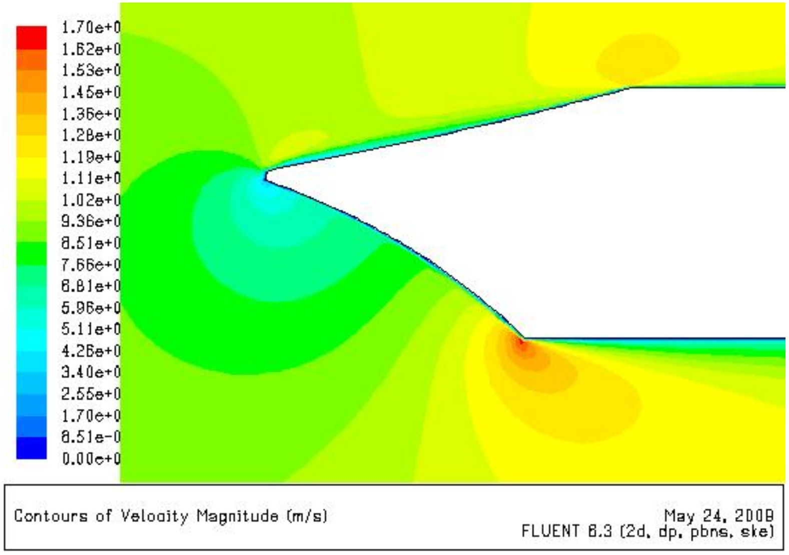


Figure 4.21 Velocity contours for Case 1 C_d-optimized truck front, Re_H = 303,957

Case 2 This case attempted to steer the genetic algorithm towards creating an improved truck front in the image of existing commercial designs. The separation between P_0 and P_3 in the *x*-coordinate was widened to 0.546 while the left boundary for the control points was kept at $x = -2$. Figure 4.22 shows the convergence plot for the genetic algorithm. Figure 4.23 shows the velocity contours for the optimized truck front. The top edge is slanted and lets the air flow over without any obstruction. The bottom edge is about 15° from the vertical and allows a stagnation point to develop just below the truck nose. This creates some boundary layer separation at P_0. The drag coefficient for this truck was 1.009, giving a 3.4% drop from the baseline case.

Although this truck front shape does not reduce the drag as substantially as does Case 1, it is more practical since it allows room for the engine and the passengers along with a slight improvement in the coefficient of drag. The Case 2 optimized truck front is also similar to exisiting commercial designs and could be easily implemented by industry.

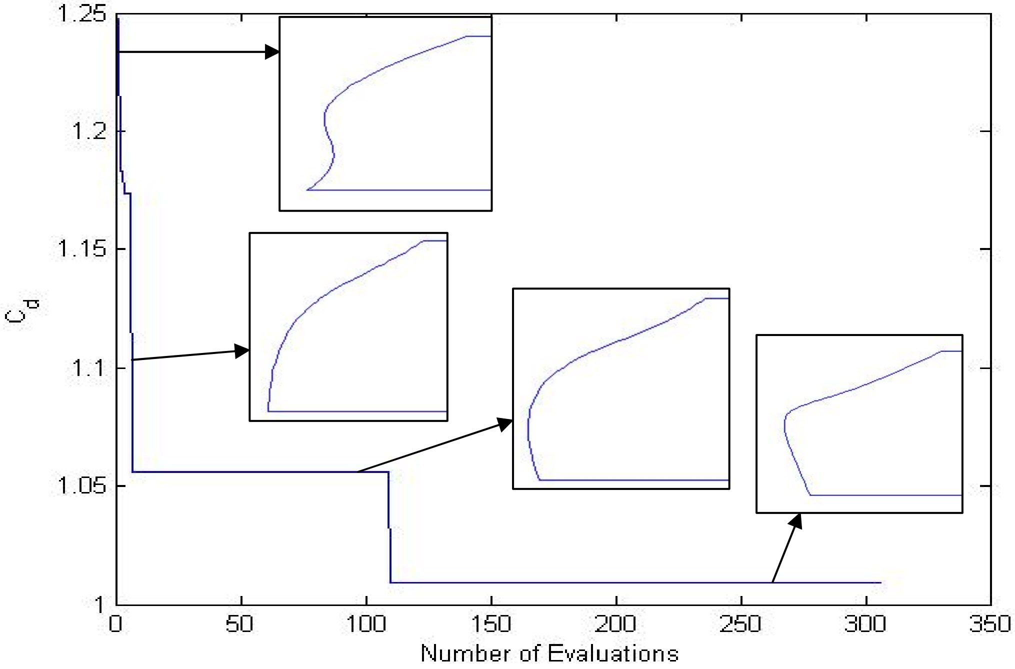


Figure 4.22 Convergence plot for Case 2 C_d-optimized truck front, $Re_H = 303,957$

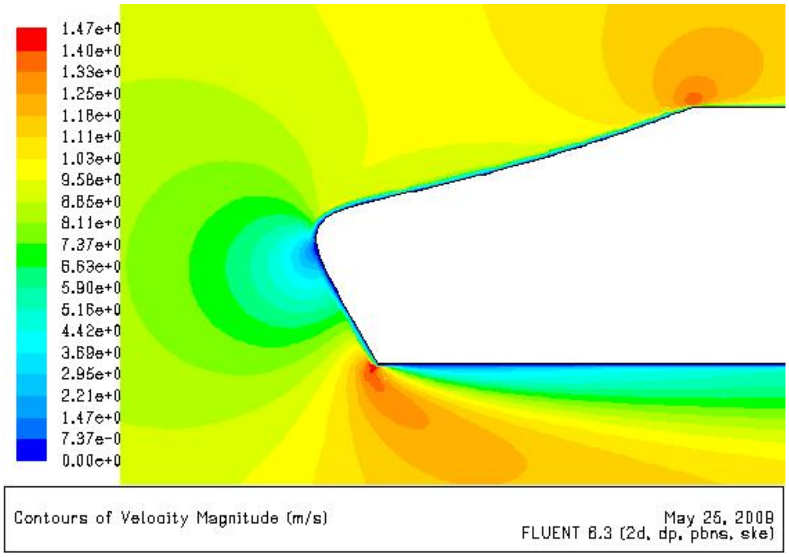


Figure 4.23 Velocity contours for the Case 2 C_d-optimized truck front, Re_H = 303,957

Case 3 This case attempted to improve on Case 2 by moving the left constraint for the control points to *x* = -3. The anchor points were kept at the same positions as in Case 2. The convergence plot for Case 3 is given in Figure 4.24. It indicates that it took about 10 generations to determine the range of optimized truck fronts. Figure 4.25 shows the velocity contours surrounding the optimized truck front. This truck front is similar to Case 1, but has a more elongated frontal nose. While both the top and bottom edges allow the air to flow smoothly, the coefficient of drag nevertheless has a high value of 1.033. This is only 1.1% less than the baseline case. Regardless, this truck front shape exhibits undesirable characteristics (lack of sufficient room for engine and passengers) that would make it hard to propose to the truck industry.

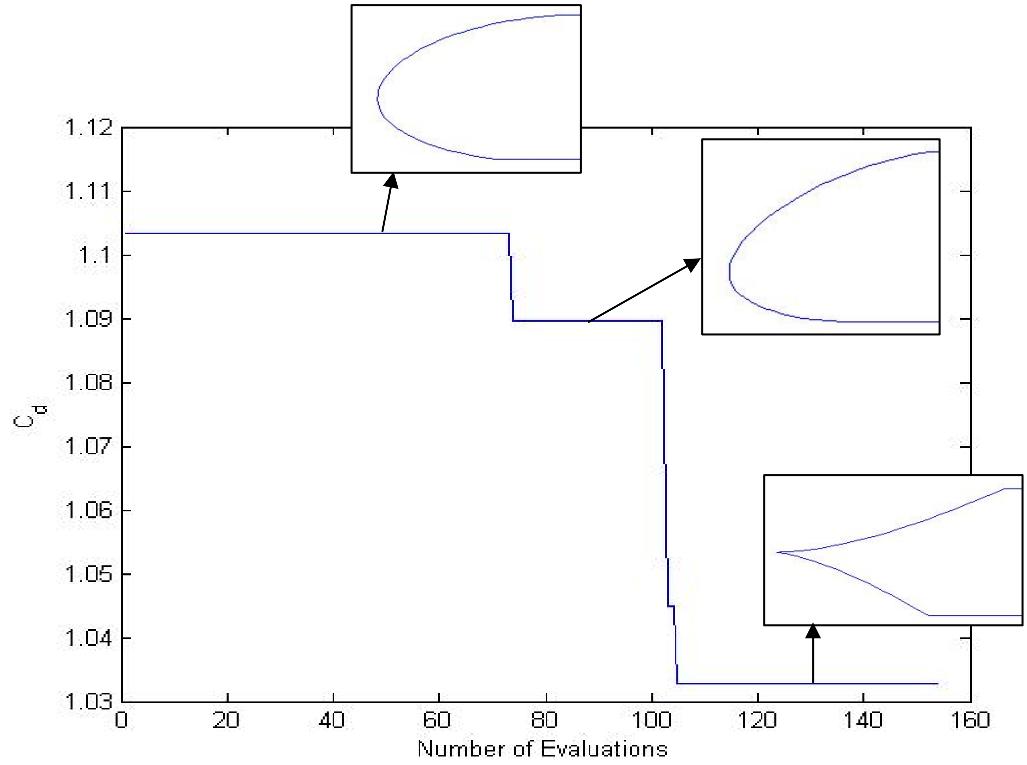


Figure 4.24 Convergence plot for Case 3 C_d-optimized truck front, Re_H = 303,957

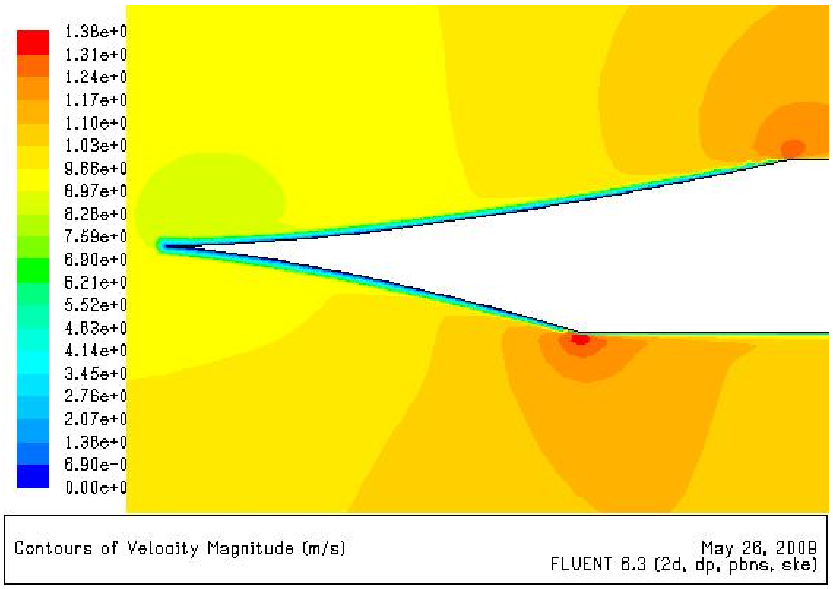


Figure 4.25 Velocity contours for the Case 3 C_d-optimized truck front, Re_H = 303,957

Case 4 This case keeps the control points' boundary at x = -3, and creates an even larger gap, 1.046, between the anchor points. This was done to try to influence the genetic algorithm to create an optimized truck front shape that would be similar to existing commercial shapes. Indeed, Figure 4.26 shows a shape that, while not consistent with trucks, looks more like a car. The shape takes a gradual positive slope at first and then becomes steeper before smoothing out to meet the top anchor point. This latter part creates a low velocity area at the front of the shape which then speeds up as it moves further downsteam. The bottom edge is horizontal and goes directly into the back part of the truck. The coefficient of drag for this optimized truck front shape was 1.056. This is 1.1% higher than that for the baseline case. While it was a useful experiment in trying different constraints, it did not result in an optimized shape with reduced drag. Figure 4.27 shows the convergence plot of the genetic algorithm for this case.

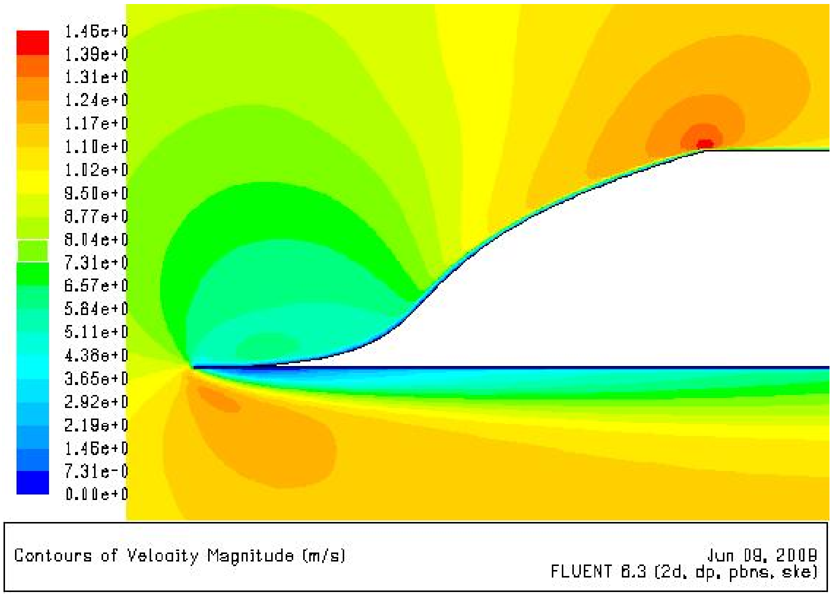


Figure 4.26 Velocity contours for Case 4 C_d-optimized truck front, Re_H = 303,957

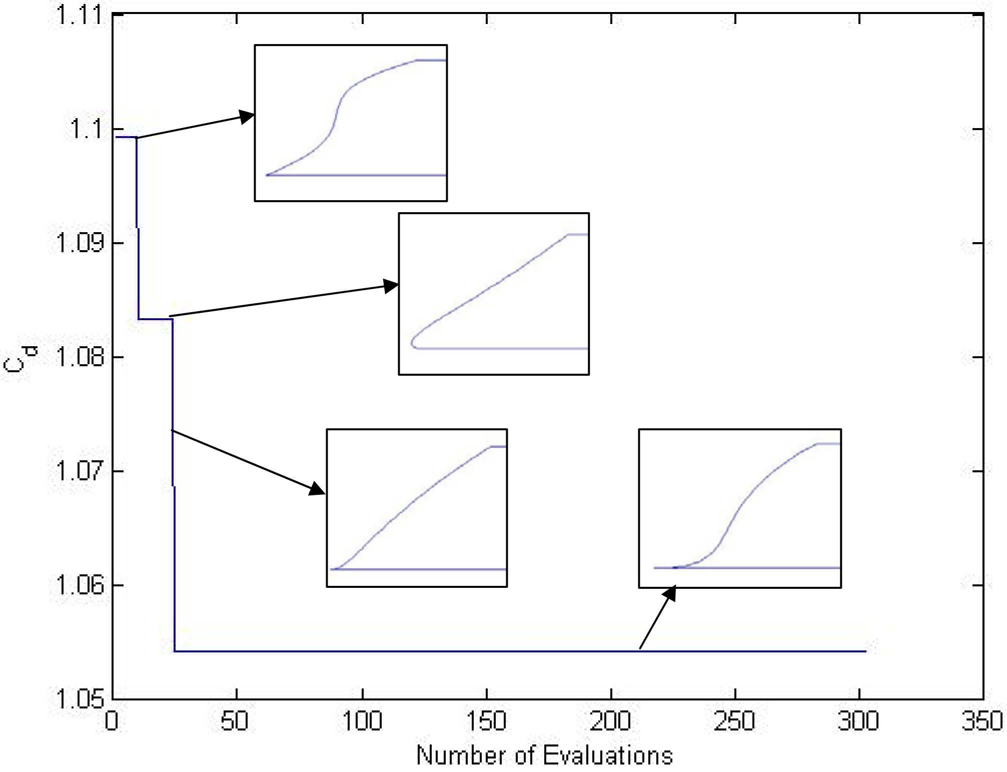


Figure 4.27 Convergence plot for Case 4 C_d-optimized truck front, Re_H = 303,957

Case 5 Finally, in Case 5, the distance between the anchor points was kept the same as in Case 4, but the left boundary for the control points was moved even further out to x = -4. This was again to determine if the distance between the anchor points allowed the algorithm to find an optimal truck front with an acceptable design. Figure 4.28 shows the optimized truck front shape and the velocity contours around it. Both the top and bottom edges do well to cut smoothly through the flow, however the top edge becomes almost vertical as it hits the top anchor point. Although this feature creates some boundary layer separation, it allows the truck front shape to be closer to existing commercial designs. The coefficient of drag was 0.712, resulting in a reduction of 31.9% from the baseline case. It appears that this truck front shape does well both to decrease the drag and keep its design within existing parameters. Unfortunately, Figure 4.28 also shows that the truck front is larger than the box trailer that the truck would be pulling. This would create a need for a lot of excess material that would increase the weight of the truck and essentially nullify some of the gains in the efficiency achieved by the decrease in drag. Figure 4.29 shows the convergence plot for the genetic algorithm for this case.

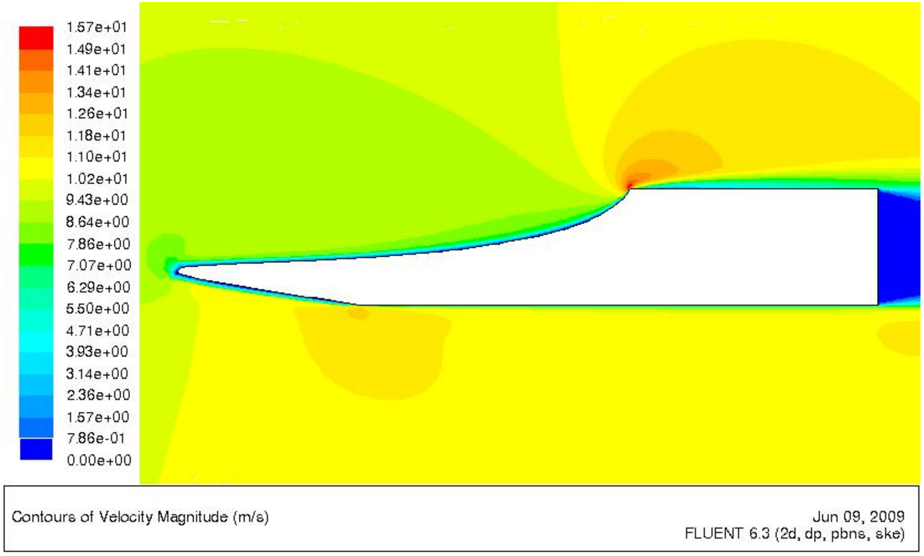


Figure 4.28 Velocity contours for Case 5 C_d-optimized truck front, $Re_H = 303,957$

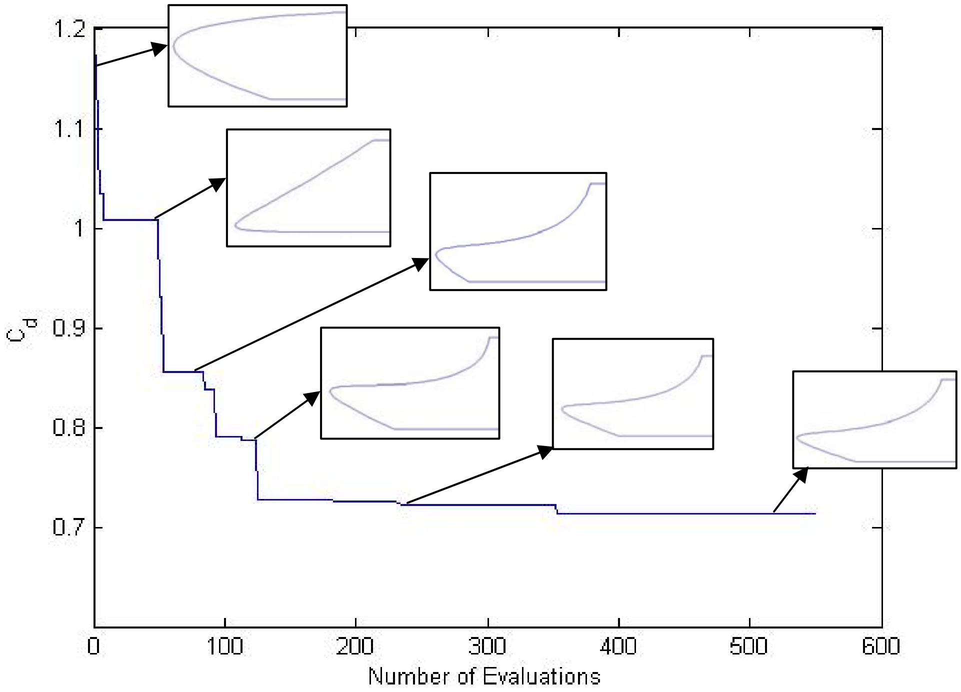


Figure 4.29 Convergence plot for Case 5 C_d-optimized truck front, $Re_H = 303,957$

4.8 Conclusions

The goal of this chapter was to find optimal airfoil shapes with maximum lift at low Reynolds numbers on the order of $Re = 10^5$ for UAV applications and to determine optimal front shapes of shipping trucks for reduced drag. However, this study also provided a lesson in understanding how to interface a genetic algorithm with a CFD solver. Specifically, a good adapted, resolved mesh journal file that works for any shape must be created before running the algorithm. A journal file that does not do this will stop the algorithm from running. The mesh should use an unstructured grid in the mesh face where the variable geometry lies. Any other face that does not interface with the geometry can use a structured grid.

The genetic algorithm also has some limitations. It searches for the optimal configuration of traits that give the best fitness value. However, the best configuration of traits usually has multiple fitness values that can make it optimal. For example, the genetic algorithm, employed for airfoil optimization, used the coefficient of lift as its fitness value; however the drag of the airfoil also matters. For the truck, the genetic algorithm used the coefficient of drag as its fitness value, but also needs to allow room for an engine and a driver. How can multiple fitness values be enforced using a genetic algorithm? Constraints must be used to push the algorithm to only consider a certain range of shapes that will uphold all the fitness values. Thus, constraints are very important for implementing the genetic algorithm.

After implementing an adapted, resolved mesh that would work for any truck front shape and constraints to address multiple fitness values, the genetic algorithm found new optimal shapes. The best shape that this author found was Case 2 as it reduced the coefficient of drag by 3.6%, but also left room for the important components of the truck. Thus, there are better truck front shapes than exisiting ones, and even better shapes could be found by adding more control points or tweaking the constraints.

Unfortunately, each case took about 2 weeks to run. This was due to the large number of iterations required to obtain a converged solution of the flow field. More computing power could make this faster in the future. Also, the use of an artificial neural network (ANN) could boost the genetic algorithm's speed. An ANN would use previously calculated fitness values and corresponding traits to train a network of nodes. The nodes would then be able to take in a set of traits and produce a fitness value without calculating the flow field. Thus, the number of needed flow field calculations needed could be reduced from 400 to 200, cutting the total genetic algorithm CPU time in half.

Chapter 5

Active Flow Control Using Synthetic Jet Actuators (SJA)

5.1 Introduction

This chapter utilizes Active Flow Control (AFC) by placement of synthetic jet actuators on the rear face of generic truck bodies to alter the wake flow pattern and consequently, reduce aerodynamic pressure drag. Studies were performed on a two-dimensional generic truck body [5], a two-dimensional D-shaped bluff body [7], and the three-dimensional Ahmed body [8, 10] described in the literature survey of chapter 2 for which the experimental data is available. There were two main parts to this research. In the first part, numerical simulations were performed for the three configurations in references [5], [7], and [10] without AFC, and were compared with the experimental data for the drag coefficient. We refer to three simulations without AFC as baseline cases. Because numerical simulations are approximations to actual fluid flow, it was important to compute the baseline results for validation to determine if this gave similar values and trends for the flow quantities as the experimental data before employing active flow control. The second part of this research was to numerically determine drag reductions with active flow control using different SJA configurations behind the three generic truck bodies and compare them with the drag reductions found experimentally.

5.2 Bluff Body Wakes

Bluff body wakes are characterized by the Kármán vortex street, named after the engineer and physicist Theodore von Kármán. A typical Kármán street behind a circular cylinder is shown in Figure 5.1 [20]. As the boundary layer from the front of the truck grows and reaches the back of the truck, it cannot stay attached to the wall and separates from the truck surface. The upper and lower surface boundary layers detach and roll up in an asymmetrical manner like a Kármán vortex street. This asymmetrical wake structure is perpetuated by a strong interdependence between the upper and lower shear layers.

In Pastoor et al. [7], the vortex formation behind a D-shaped bluff body is described. Figure 5.2, from Reference [7], shows vortex A being created behind the D-shaped body by a roll-up of the upper shear layer. This in turn induces the creation of a new vortex by separation of the lower boundary layer; it is denoted as B. Once vortex B grows in size and moves downstream, it triggers the creation of a new vortex from the upper shear layer. These strong vortices in the near wake result in a small deadwater region, or area of stagnant flow, and strongly bend the shear layers. This recirculating flow results in low pressure at the base of the body, which contributes to high pressure drag. Pressure drag is a difference between the streamwise force due to pressure acting on the front and back part of the bluff body.

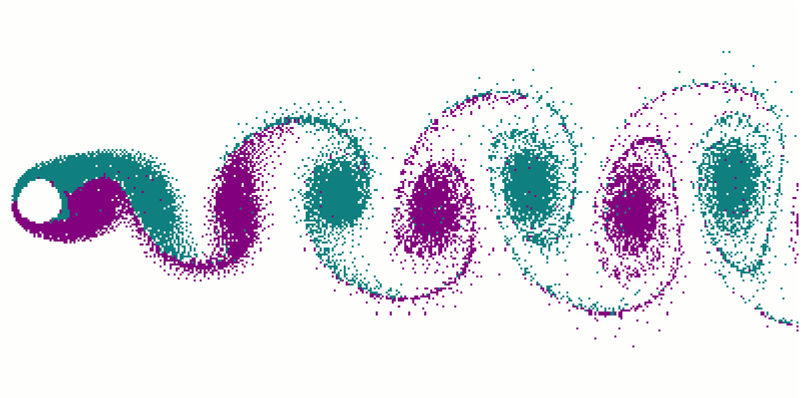


Figure 5.1 Kármán vortex street behind a cylinder [20]

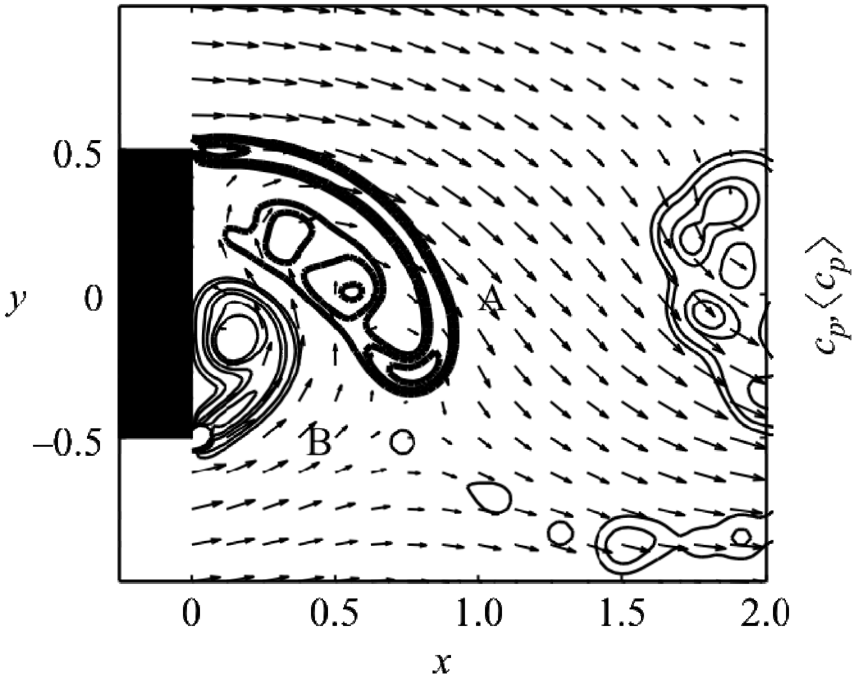


Figure 5.2 Asymmetrical vortex shedding from a D-shaped bluff body [7]

In order to reduce drag with active flow control, it must change the wake structure behind the bluff body such that the base pressure increases. This can be achieved by energizing the upper and lower boundary layers to delay vortex formation. The aim is to synchronize the vortex shedding from the boundary layers farther away from the truck's base. This will create a larger deadwater region which will increase the pressure at the base and thereby reduce drag. Figure 5.3 illustrates this new wake structure by applying AFC.

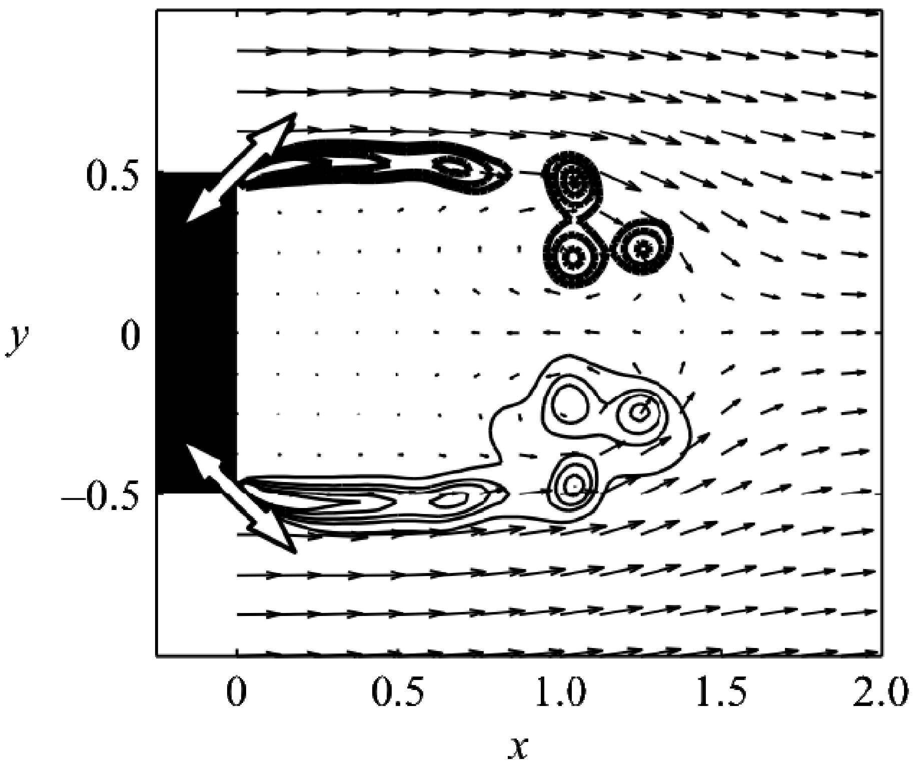


Figure 5.3 Symmetrical vortex shedding behind the D-shaped bluff body as a result of active flow control [7]

5.3 Synthetic Jet Actuators (SJAs)

While the use of passive flow control devices for drag reduction (vortex generators, microramps, LEBU, etc.) has been intensively studied in the past five decades, active flow control methods, other than injection and suction, have been a subject of intense study only during the past decade. Steady-state blowing or suction into the boundary layer have been tried dating back to World War II, but, due to the added penalty of weight from additional ducting systems and maintenance issues, were never implemented on a commercial aircraft. However, recent development in AFC devices such as oscialltory jets, microjets, splash jets, and pulsed jets have shown possibilities for implementation of these devices on commercial aircraft in the not too distant future.

In this thesis, we consider a relatively new active flow control device called an oscillatory jet or synthetic jet actuator (SJA). This device creates a net-zero-mass-flux momentum impulse into the flow. The device is light weight, compact, and low cost. It can be easily installed on aircraft wings or trucks without the need for any fluidic plumbing. Futhermore, the SJA has the potential of being significantly more efficient than its steady-state counterparts.

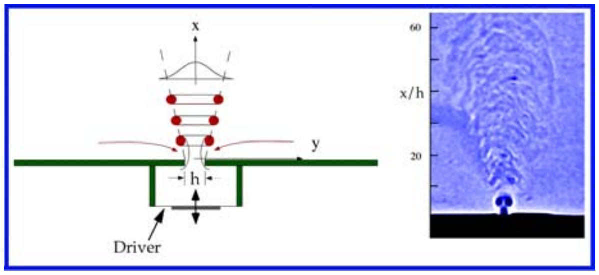


Figure 5.4 Schematic of SJA operation [21]

A simple schematic of the operation of the SJA is shown in Figure 5.4. Figure 5.5 shows the detailed schematic of the operation of an SJA. The actuating diaphragm or membrane is made from carbon fiber. The diaphragm is attached to a hollow cylinder around which an electric copper coil is wrapped. The diaphragm and electric coil cylinder assembly is surrounded by a magnet with a small gap. When a voltage on the order of 6 V amplitude is applied to the electric coil, current flows through the coil. In the presence of the magnetic field of the surrounding magnet, an electromotive force is induced which moves the cylinder/diaphragm assembly. The diaphragm follows a sinusoidal motion at the frequency of the applied AC voltage. As the diaphragm oscillates, the air is pushed out and sucked in through a small slit at the front of the SJA assembly as shown in Figure 5.5. A typical SJA and its hardware assembly are shown in Figure 5.6. The jet can issue from both the front and back parts of the assembly. These SJAs can be easily flush-integrated with the surface of a wing or truck back.

The performance of an SJA is characterized by two key non-dimensional parameters. The Strouhal number, *St*, compares the SJA operating frequency to the typical time it takes the fluid to advect through the boundary layer:

$$St = \frac{fH}{U_\infty}, \tag{12}$$

where f is the operating frequency, H is the height of the truck, and U_∞ is the free stream velocity. The momentum coefficient, c_μ, specifies the momentum the SJA imparts into the flow. It also provides a measure of the cost of actuation:

$$c_\mu = 2\frac{S}{H}\frac{q_A^2}{U_\infty^2}, \tag{13}$$

where S is the width of the SJA slit and q_A is the root mean square value of $v_{SJA} = v_0 sin(2\pi ft)$; v_0 being the amplitude of the oscillatory velocity of the SJA.

Most importantly, the power costs of the SJAs must be quantified to show that deployment is not unreasonably expensive. Crowther and Gomes [19] have developed the following equation for the required fluid power for a flow control application:

$$W_F = \frac{\pi}{12}\frac{V_R^3 \Delta}{\lambda}\frac{1}{2}\rho s_A \delta U_\infty^3, \tag{14}$$

where V_R is the ratio of v_0 to U_∞, Δ is the ratio of S to the boundary layer height δ, λ is the ratio of SJA spanwise spacing to S, and s_A is the spanwise extent of the SJA array. The boundary layer height is estimated by a zero pressure gradient laminar or turbulent flat plate model given as:

$$\delta_{lam} = 4.91\left(\frac{\nu}{U_\infty}\right)^{1/2} x^{1/2}, \quad \delta_{turb} = 0.383\left(\frac{\nu}{U_\infty}\right)^{1/5} x^{4/2}. \tag{15}$$

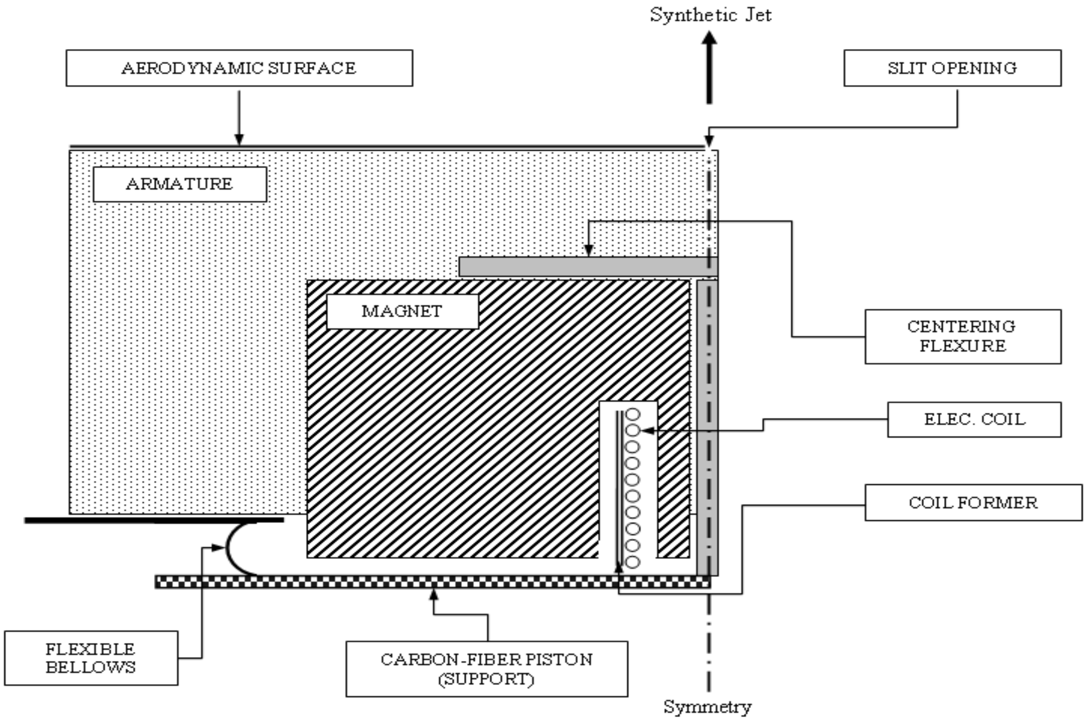


Figure 5.5 Detailed schematic of the operation of the SJA [21]

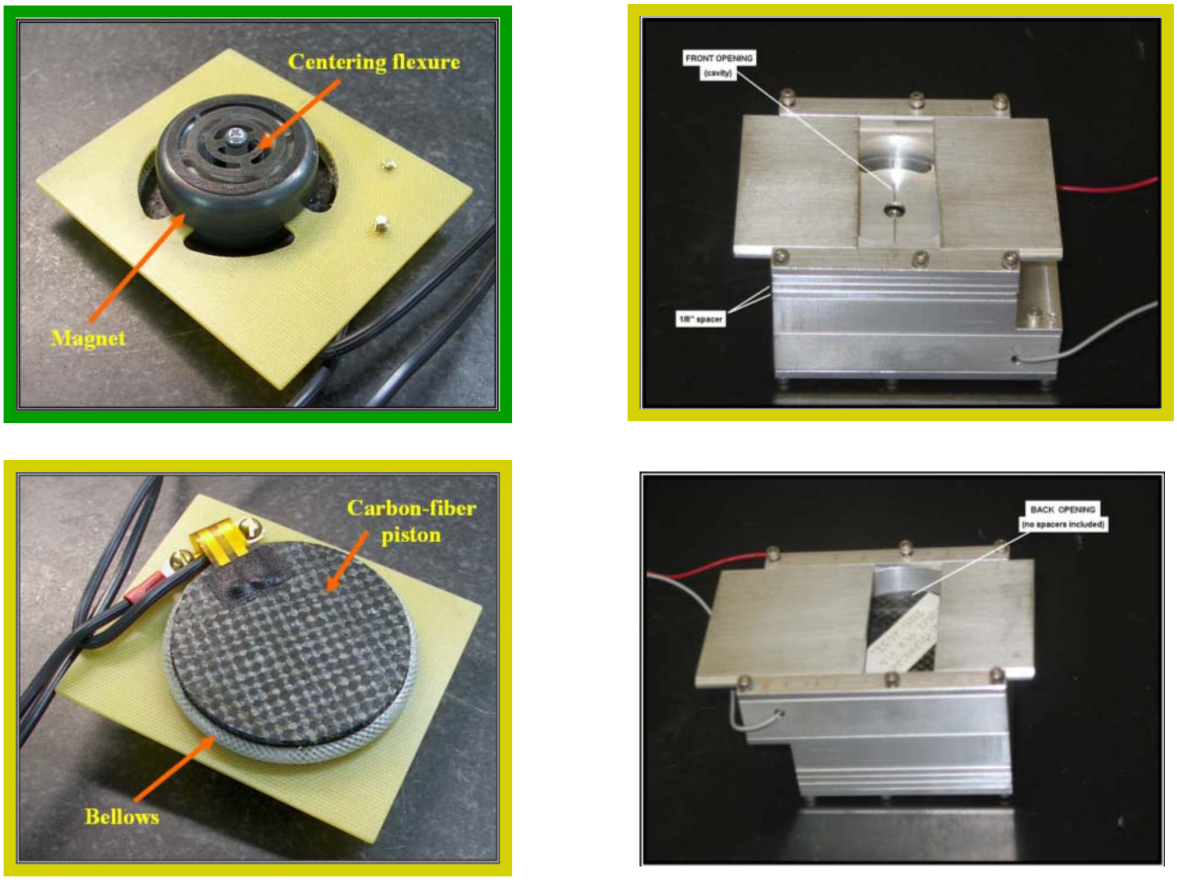


Figure 5.6 Typical parts of the SJA (on left) and SJA assembly (on right) [21]

5.4 Generic Truck Body

5.4.1 Introduction

Numerical simulations were performed on a two-dimensional generic truck body described in section 2.1. The computed baseline flow field was compared to the experimental data to determine an adaptive grid distribution and turbulence model which provided the closest agreement between the two. The grid was sufficiently dense so that the solution was almost grid independent. Several turbulence models were investigated namely the Spalart-Allmaras (SA), k-ε, and realizable k-ε. It was found that the realizable k-ε model gave the best results when compared with the experimental data. After the validation of the baseline case (without AFC), active flow control was employed using three different configurations of SJAs on the base of the truck. Finally, the relative effectiveness of these configurations was determined by comparing the computed drag coefficient. The wake structure and the pressure distributions at the truck base were analyzed.

5.4.2 Computational Solution Procedure

Geometry Modeling and Grid Generation The generic truck body geometry from reference [5] was created in GAMBIT as shown in Figure 5.7. Figure 5.8, Figure 5.9, and Figure 5.10 show the complete mesh, zoomed-in view of the mesh near the truck, and the adapted mesh in the synthetic jet regions respetively. In Figure 5.8, the computational domain has a semi-circular farfield with a radius of 12.5L; the downstream farfield is at 20L from the base of the truck. These farfield faces have rectangular cells which are used to create a structured mesh. The mesh decreases in cell size as it moves towards the truck geometry. The mesh surrounding the truck, shown in Figure 5.9, was adapted for better flow field resolution. The resulting adapting mesh had a node/cell/face count of 28,185/56,936/28,751.

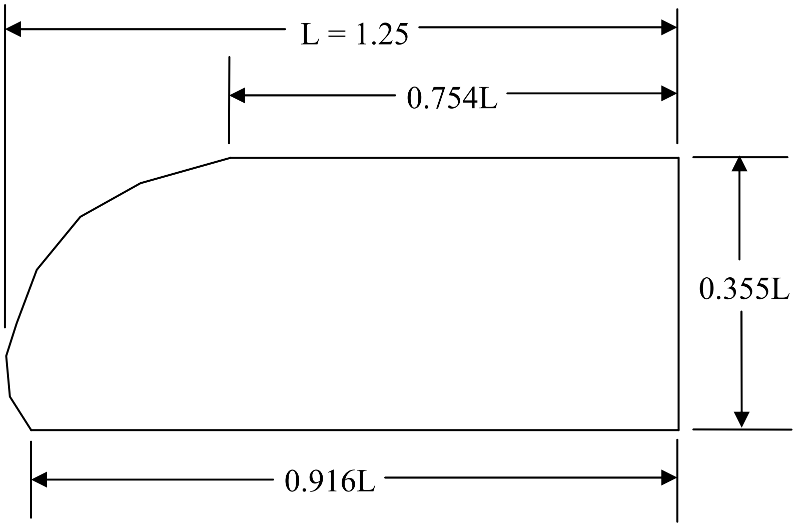


Figure 5.7 Generic truck dimensions [5]

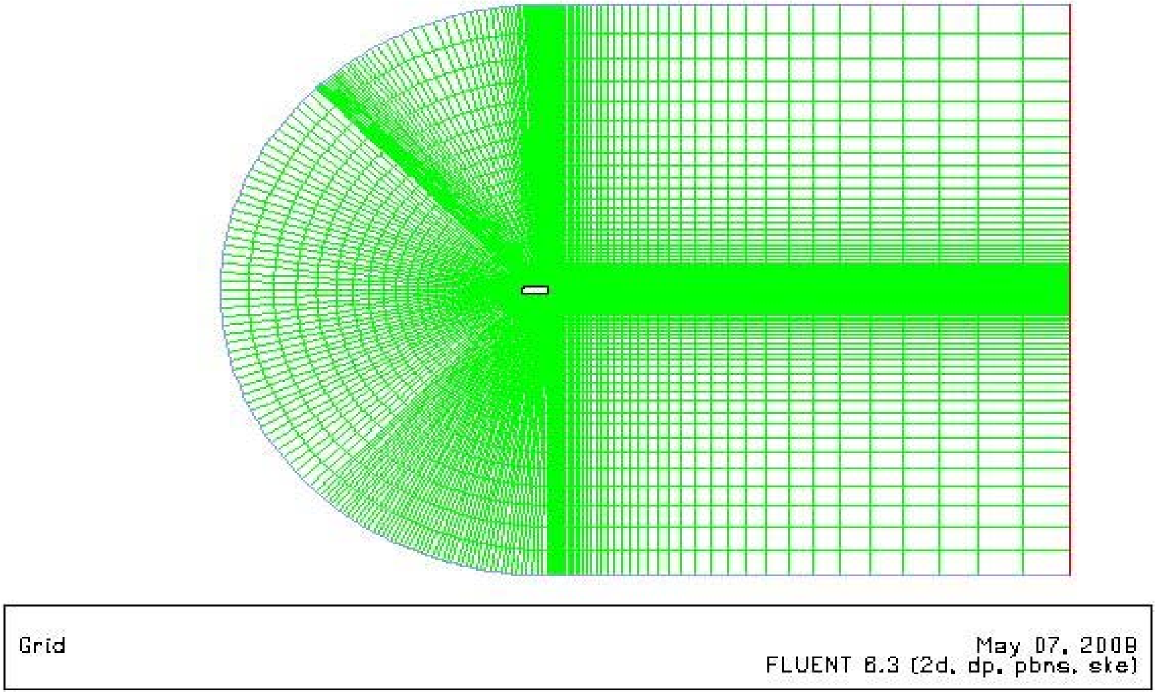


Figure 5.8 Generic truck body: complete computational domain with structured mesh

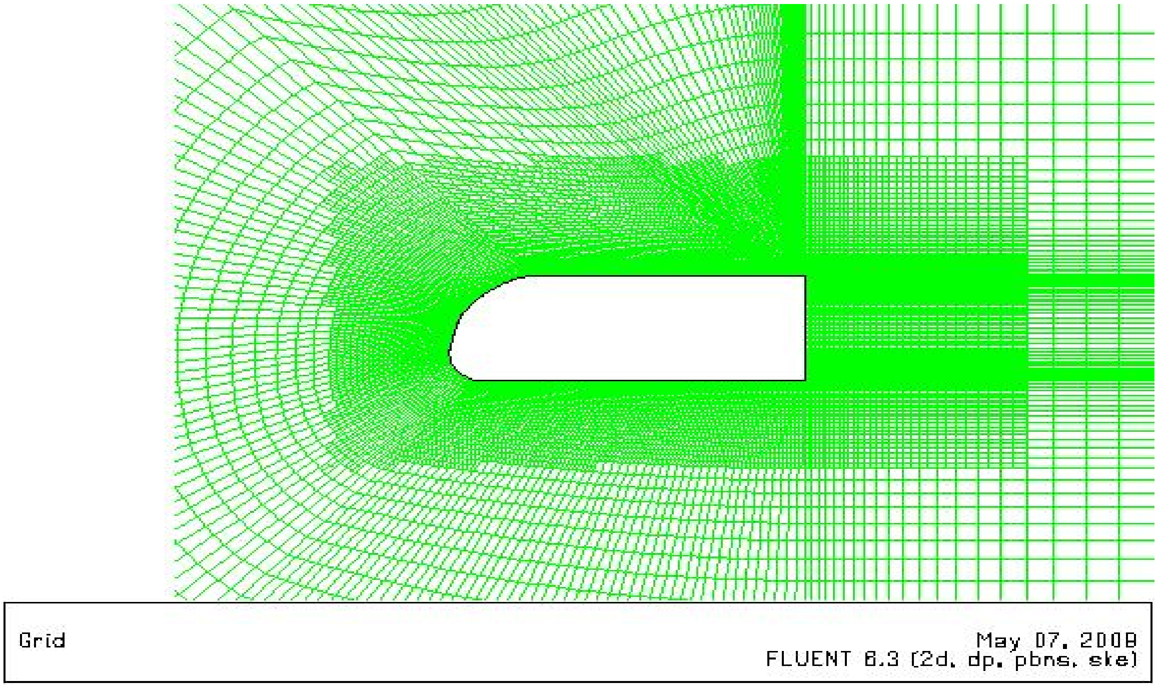


Figure 5.9 Generic truck body with zoomed-in mesh near the body

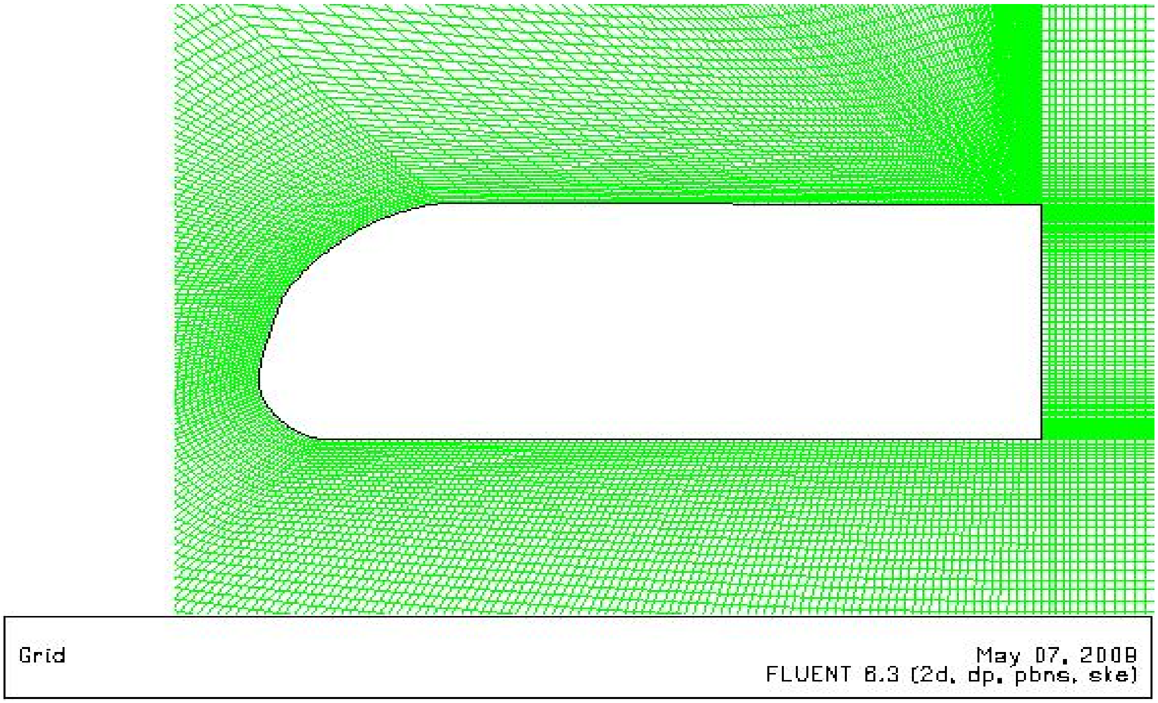


Figure 5.10 Generic truck body with zoomed-in adapted mesh very close to the body and synthetic jet regions

Several SJAs, each with 1.7 mm in slit width (same as in the experiment of Seifert et al. [5]), were placed on the back of the generic truck at strategic locations. Figure 5.11 shows the SJA configuration in which two SJAs were placed near the top and bottom of the rear of the truck in the direction of the free stream. Figure 5.12 shows the SJA configuration in which two SJAs operate at an angle of 45° to the free stream direction. Figure 5.13 shows the the third configuration in which two SJAs are placed at the back of the truck as in the first configuration of Figure 5.11 with an additional SJA placed at the top surface of the truck near the boundary layer separation point.

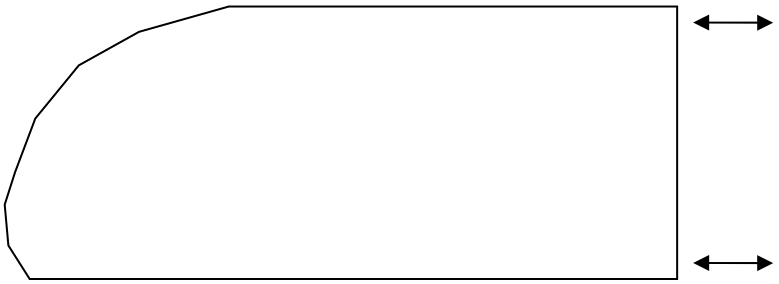


Figure 5.11 Configuration 1: Two SJAs at the rear face of the truck in the direction of the free stream

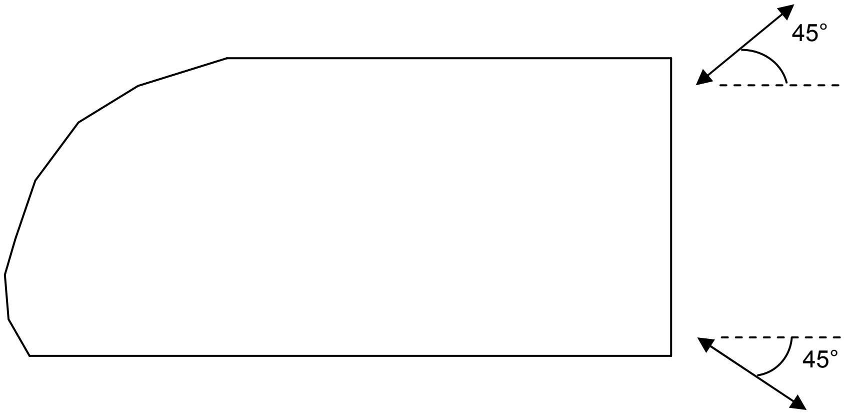


Figure 5.12 Configuration 2: Two SJAs at the rear face of the truck at an inclination of 45° to the free stream

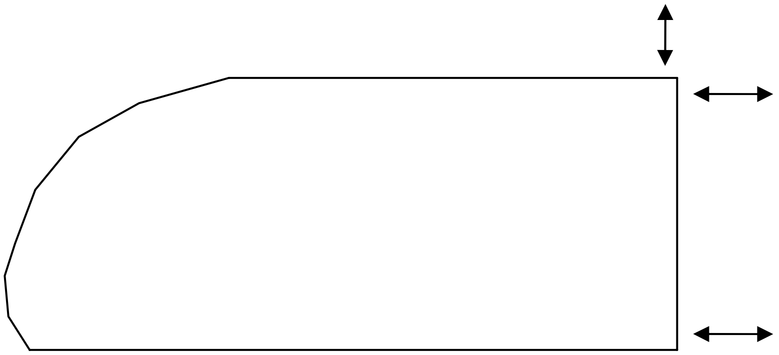


Figure 5.13 Configuration 3: Two SJAs at the rear face of the truck in the direction of the free stream and one SJA on the top surface of the truck

CFD Solution The commercial CFD solver FLUENT was employed to compute the flow fields and the coefficient of drag. Baseline (without AFC) and flow fields with active flow control were computed at three free stream velocities: 10, 20, and 30 m/s corresponding to Reynolds numbers of 3.08×10^5, 6.16×10^5, and 9.24×10^5

respectively based on the height of the truck as the characteristic length. The free stream temperature and static pressure values were taken to be at sea level conditions; that is 288.5 K and 101325 Pa respectively. The density of air ϱ = 1.225 kg/m^3 and the dynamic viscosity of air μ =1.849x10^{-5} kg/m-s were used. The flow near the front of the truck was considered laminar while the wake flow behind the truck was considered turbulent. The realizable k-ε turbulence model was employed. For computations of flow fields with active flow control, the boundary condition $v_{SJA} = v_0 sin(2\pi ft)$ for SJAs was employed over the slot width with f = 100 Hz and $v_0 = 0.5U_\infty$.

5.4.3 Results and Analysis

Computed and experimental results for baseline (without AFC) flow field results are given in Table 5.1; they are also shown in Figure 5.14 in graphical form. Computed experimental results show different trends as the Reynolds number increases. The experiment shows that the coefficient of drag C_d increases as the Reynolds number increases while the computations show that the coefficient of drag C_d decreases as the Reynolds number increases. However, the computed and experimental values of C_d differ at most by 6.6%; in this sense the computed and experimental values can be considered close enough. We believe that the numerical results are accurate. The coefficient of drag varies with the drag force in the numerator and the square of the free stream velocity in the denominator. While the drag force increases with an increase in the free stream velocity, the velocity in the denominator increases quadratically. Thus, the coefficient of drag should decrease with an increase in the Reynolds number (for the range of Reynolds numbers considered). This result agrees with the results reported in the literature.

For the baseline case, computed velocity contours behind the truck are shown in Figure 5.16, Figure 5.17, and Figure 5.18 at Reynolds numbers 3.08 x 10^5, 6.16 x 10^5, and 9.24 x 10^5 respectively. Each figure shows the recircultaing flow in the base flow region which occurs when the boundary layers from the top and bottom surface of the truck separate from the truck body. Figure 5.16 shows the vortex formation occurring near the upper boundary layer region as well as the vortex created by the separation of the lower boundary layer which moves up and away from the base. As the time progresses, the new vortex formation shifts to the region near the bottom boundary layer and the upper vortex moves downward and away from the base. This process repeats itself and forms the Kármán vortex street. The asymmetrical shedding of vortices creates a low pressure region at the base of the truck. The difference between the pressures at the front of the truck and the back of the truck results in aerodynamic pressure drag. The goal of active flow control is to reduce this pressure drag.

Table 5.1 Experimental versus computed C_d for the generic truck body for the baseline case (without AFC)

Reynolds Number	**Experimental C_d**	**Computed C_d**	**% Difference w.r.t. experimental C_d**
3.08×10^5	0.98	1.045	6.6
6.16×10^5	0.98	0.993	1.3
9.24×10^5	1.02	0.978	4.1

Computed active flow control results with two SJAs as in configuration 1 (Figure 5.11) are shown in Table 5.2. A reduction in drag can be observed with respect to baseline values. The reduction in the coefficient of drag increases from 8.94% to 15.62% as the Reynolds number increases. The velocity contour plots near the base of the truck for each Reynolds number are shown in Figure 5.19, Figure 5.25, and Figure 5.31. It can be observed that the recirculating flow region has become almost symmetric with a large deadwater region (indicated in dark blue). The upper and lower boundary layers are rolling up and creating vortices further away from the base than in the baseline flow. The vortices are also being shed symmetrically. This feature of the flow increases the pressure at the truck base. Figure 5.20, Figure 5.26, and Figure 5.32 show this increase in the pressure at the base. These figures show the baseline pressure variation at the truck base in the blue-dashed line and the active flow control pressure variation at the truck base in the red solid line. The base pressure with AFC is more uniform and larger than the base pressure for the baseline case. The greater baseline pressure with active flow control results in a decrease in the drag.

Table 5.2 Computed active flow control results for the generic truck body with two SJAs in configuration 1 (Figure 5.11)

Reynolds Number	Baseline C_d (without AFC)	C_d with AFC SJA config. 1	% Change in C_d w.r.t. baseline case
3.08×10^5	1.045	0.951	8.94
6.16×10^5	0.993	0.860	13.41
9.24×10^5	0.978	0.825	15.62

Computed active flow control results with two SJAs in configuration 2 (Figure 5.12) are shown in Table 5.3. Again, reduction in drag was obtained at all three Reynolds numbers. The reduction in coefficient of drag from 8.75% to 15.79% was obtained as the Reynolds number increased. The velocity contour plots at the truck base for each Reynolds number are shown in Figure 5.21, Figure 5.27, and Figure 5.33. As in the case of configuration 1 (Figure 5.11) with the two SJA setup, the recirculating flow has become almost symmetric with a larger deadwater region (shown in dark blue). While the upper and lower boundary layers continue to roll up and create vortices, this occurs further away from the truck base than in the case with baseline flow where the asymmetric Kármán vortex street develops close to the base flow region. In this case, vortices are again being shed almost symmetrically. Again, the pressure in the base region increases. Figure 5.22, Figure 5.28, and Figure 5.34 show this increase in pressure compared to the baseline case (without AFC). Again, the base pressure with AFC is more uniform compared to the baseline base pressure. This increase in pressure with active flow control results in reduced aerodynamic pressure drag.

Table 5.3 Computed active flow control results for the generic truck body with two SJAs in configuration 2 (Figure 5.12)

Reynolds Number	Baseline C_d (without AFC)	C_d with AFC SJA config. 2	% Change in C_d w.r.t. baseline case
3.08×10^5	1.045	0.953	8.75
6.16×10^5	0.993	0.856	13.78
9.24×10^5	0.978	0.823	15.79

Computed active flow control results with three SJAs in configuration 3 (Figure 5.13) are shown in Table 5.4. Again, reduction in drag is obtained at all three Reynolds numbers. The reduction in coefficient of drag from 6.35% to 14.16% was obtained as the Reynolds number increased. The velocity contour plots at the truck base for each Reynolds number are shown in Figure 5.23, Figure 5.29, and Figure 5.35. Again, the recirculating flow region has become almost symmetric with a larger deadwater region (shown in dark blue). While the upper and lower boundary layers continue to roll up and create vortices, this occurs further away from the truck base than is the case with the baseline flow. In this case again, the vortices are being shed symmetrically. Again, the pressure in the truck base region increases. Figure 5.24, Figure 5.30, and Figure 5.36 show this increase in pressure compared to the baseline case (without AFC). Again, the base pressure with AFC is more uniform compared to the baseline base pressure. This increase in pressure with active flow control results in reduced aerodynamic pressure drag.

Using Crowther and Gomes' power calculation, each SJA in each configuration would require 0.0058%, 0.046%, and 0.156% of the original propulsive power at Reynolds numbers of 3.08×10^5, 6.16×10^5, and 9.24×10^5 respectively.

Table 5.4 Computed active flow control results for the generic truck body with three SJAs in configuration 3 (Figure 5.13)

Reynolds Number	Baseline C_d (without AFC)	C_d with AFC SJA config. 3	% Change in C_d w.r.t. baseline case
3.08×10^5	1.045	0.978	6.35
6.16×10^5	0.993	0.868	12.63
9.24×10^5	0.978	0.839	14.16

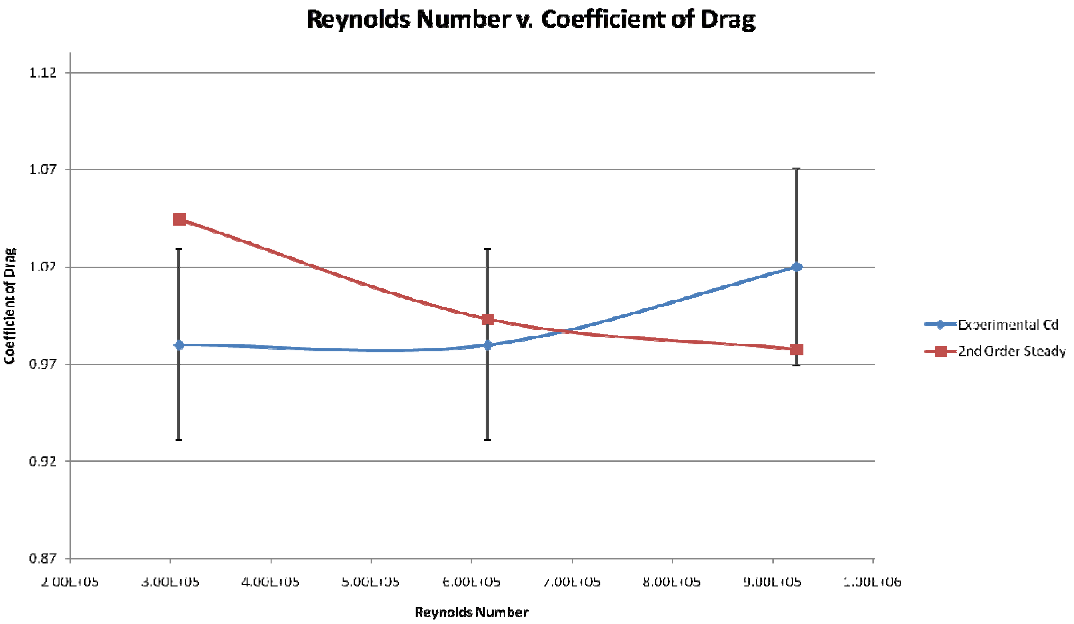


Figure 5.14 Experimental C_d versus computed C_d for the generic truck body for the baseline case (without AFC)

Figure 5.15 shows a comparison of the coefficient of drag for active flow control with three different SJA configurations versus the baseline coefficient of drag. The second SJA configuration (Figure 5.12), with two SJAs at an inclination of 45° with respect to the free stream placed at the top and bottom of the rear face of the truck, gives the largest drag reductions followed closely by the first SJA configuration (Figure 5.12) with two SJAs placed at the top and bottom corners of the truck parallel to the free stream. The third configuration with three SJAs (Figure 5.13) also reduces the drag. The largest drag reduction for each configuration occurs at Reynolds number 9.24 x 10^5 and the smallest drag reduction occurs at Reynolds number 3.08 x 10^5.

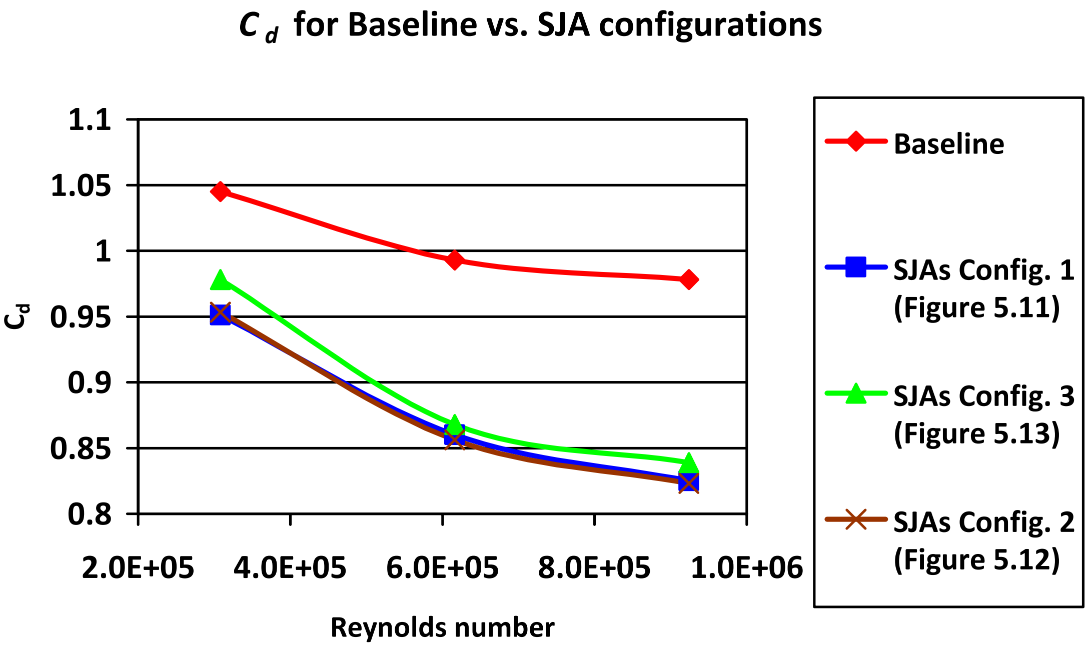


Figure 5.15 Computed C_d with active flow control for the generic truck body for three SJA configurations: config. 1 (Figure 5.11), config. 2 (Figure 5.12), and config. 3 (Figure 5.13)

Baseline

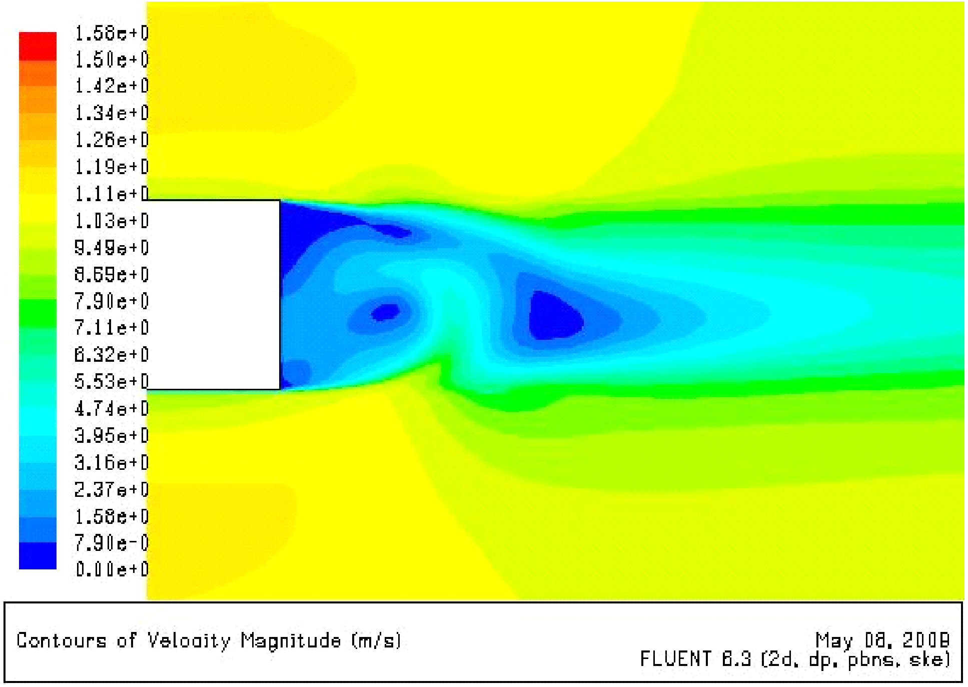


Figure 5.16 Computed velocity contours for the baseline flow (without AFC) at *Re* = 3.08 x 10^5 for the generic truck body

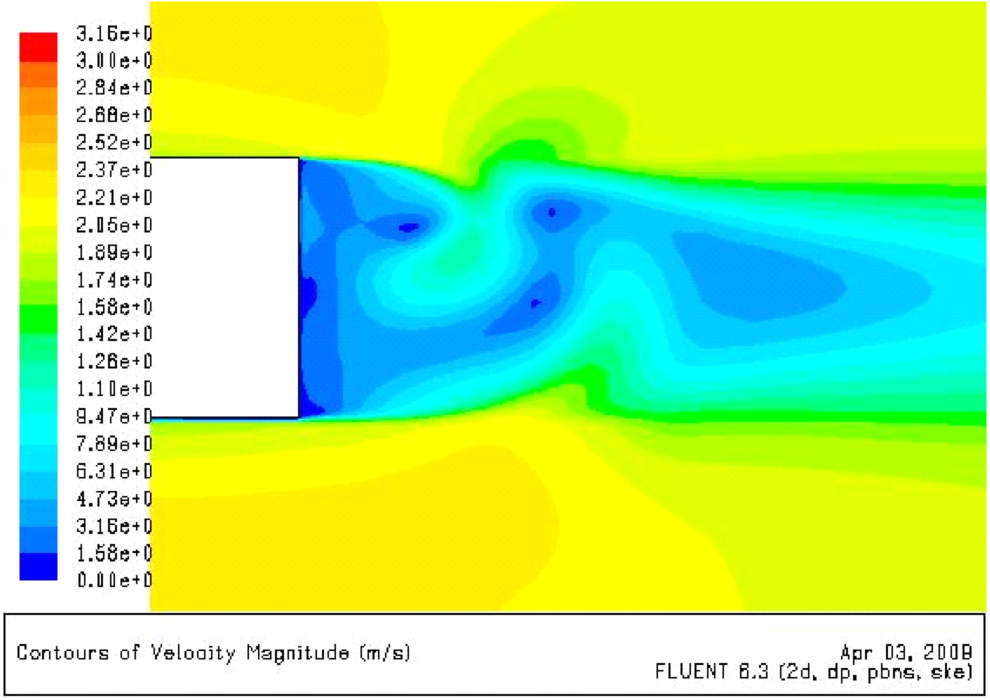


Figure 5.17 Computed velocity contours for the baseline flow (without AFC) at *Re* = 6.16 x 10^5 for the generic truck body

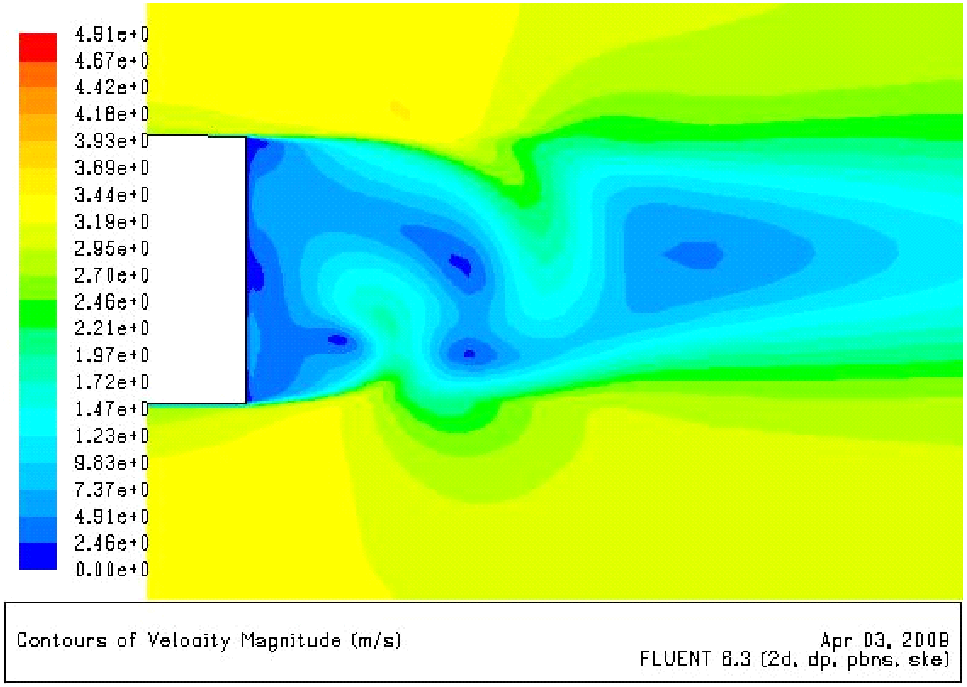


Figure 5.18 Computed velocity contours for the baseline flow (without AFC) at *Re* = 9.24 x 10^5 for the generic truck body

Active Flow Control

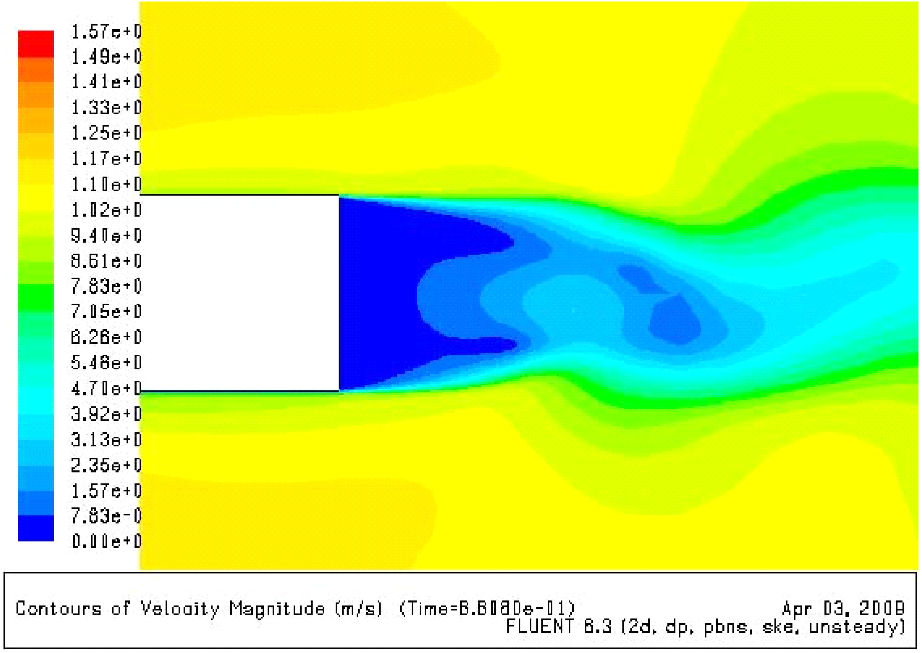


Figure 5.19 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at *Re* = 3.08 x 10^5

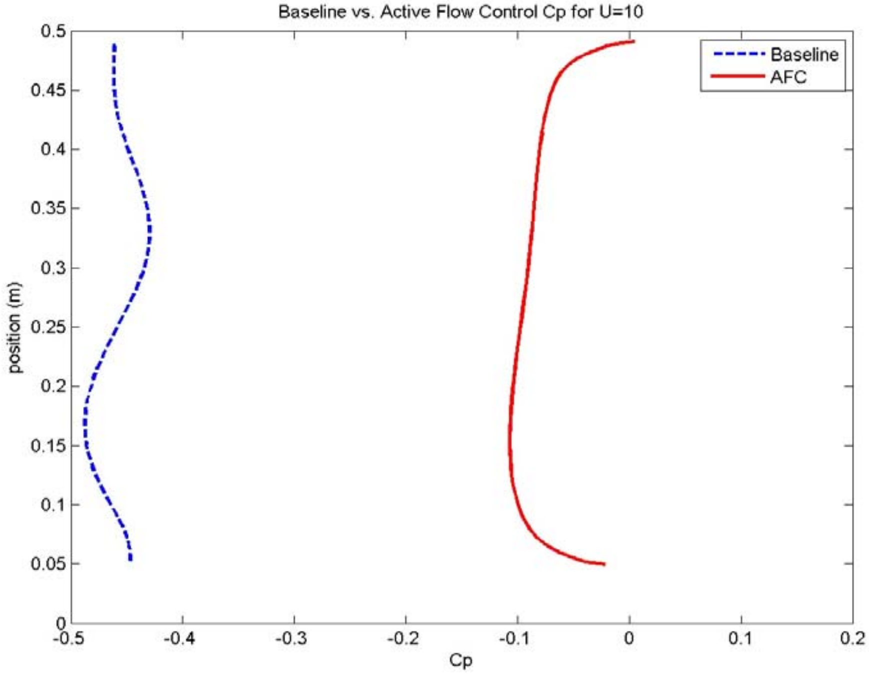


Figure 5.20 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at *Re* = 3.08 x 10^5

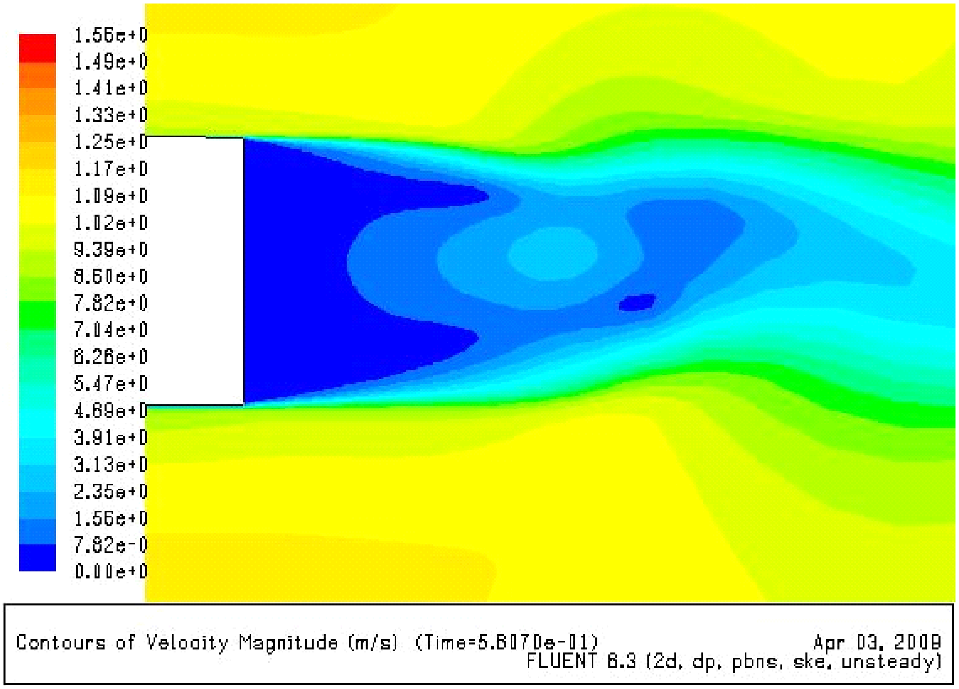


Figure 5.21 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at *Re* = 3.08 x 10^5

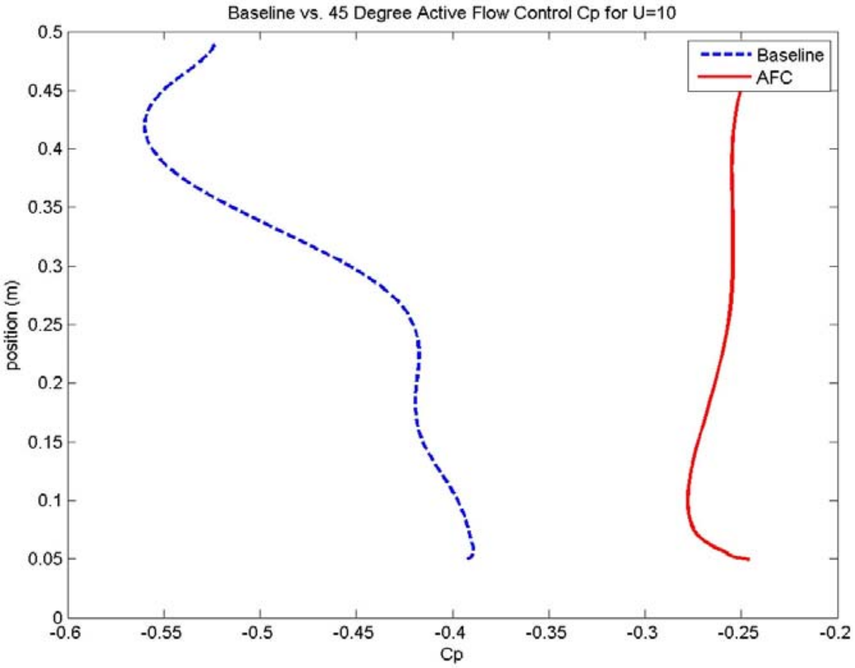


Figure 5.22 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at $Re = 3.08 \times 10^5$

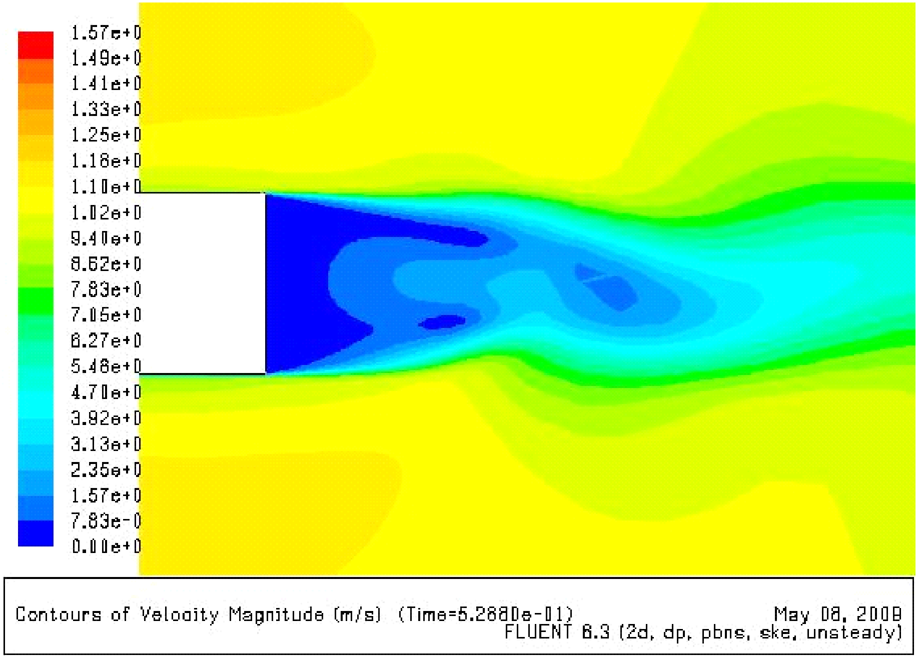


Figure 5.23 Computed velocity contours for flow with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at $Re = 3.08 \times 10^5$

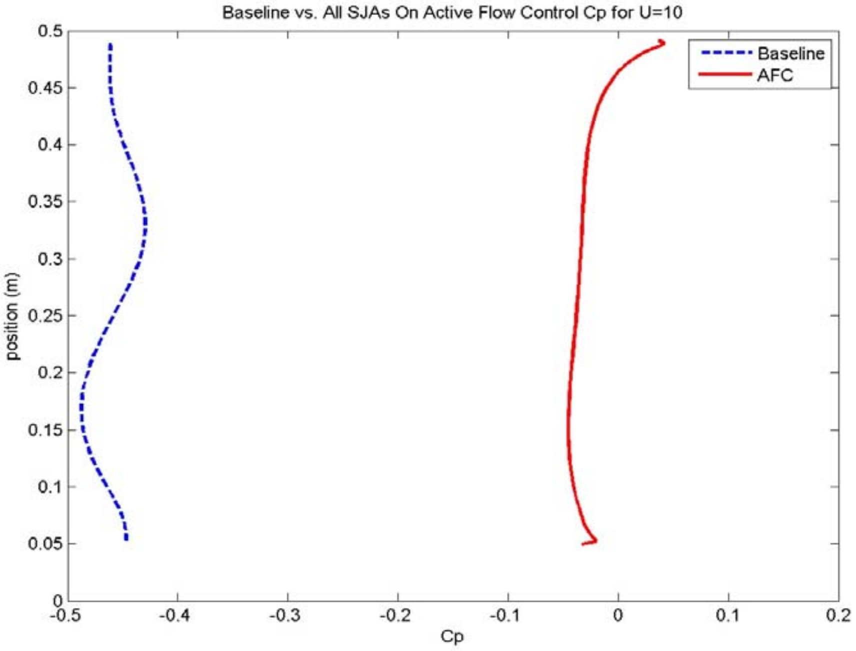


Figure 5.24 Baseline C_p versus C_p with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at *Re* = 3.08 x 10^5

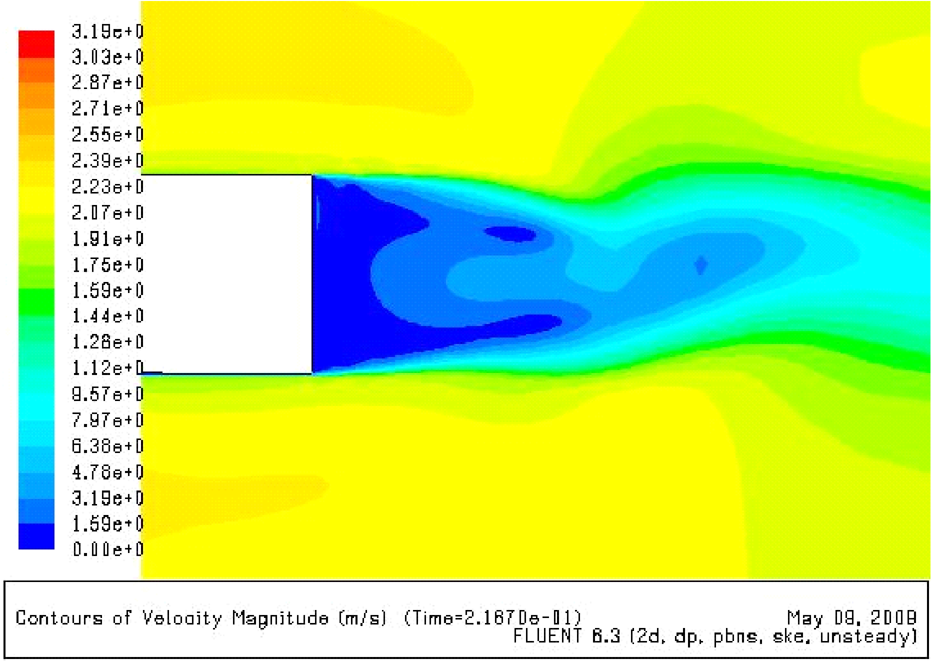


Figure 5.25 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at *Re* = 6.16 x 10^5

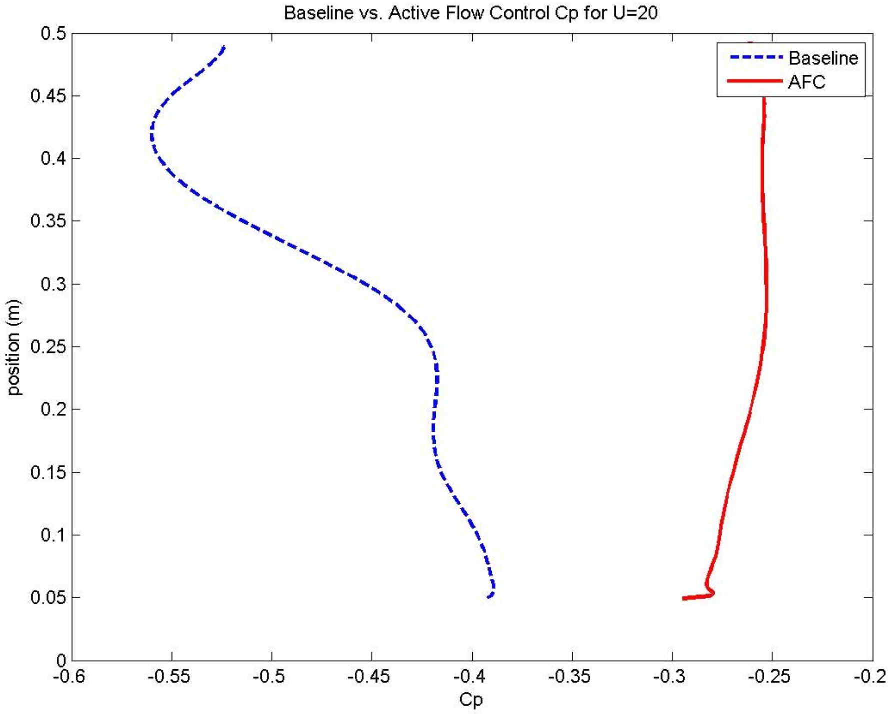


Figure 5.26 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at *Re* = 6.16 x 10^5

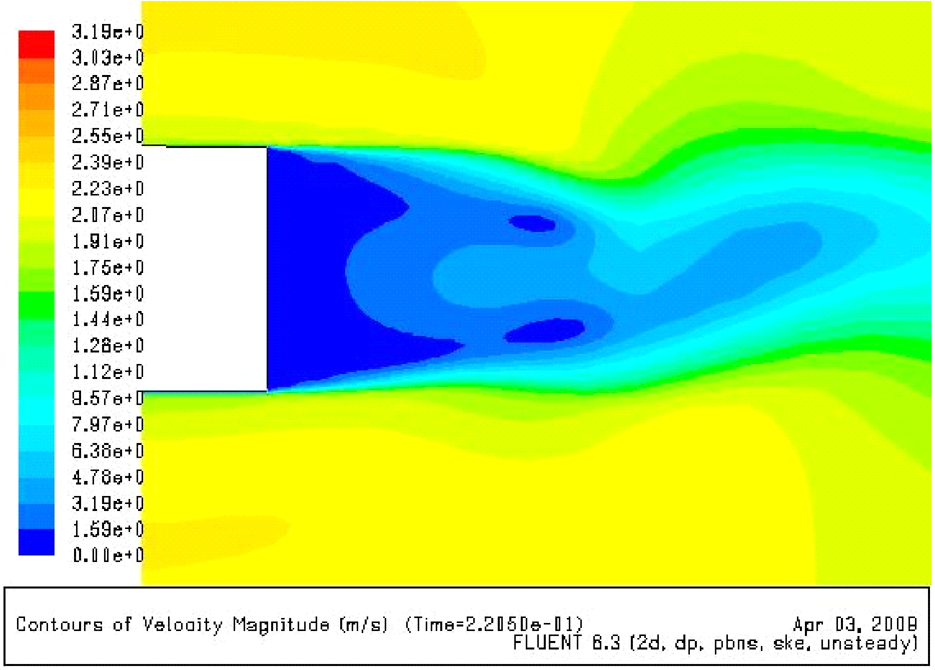


Figure 5.27 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at *Re* = 6.16 x 10^5

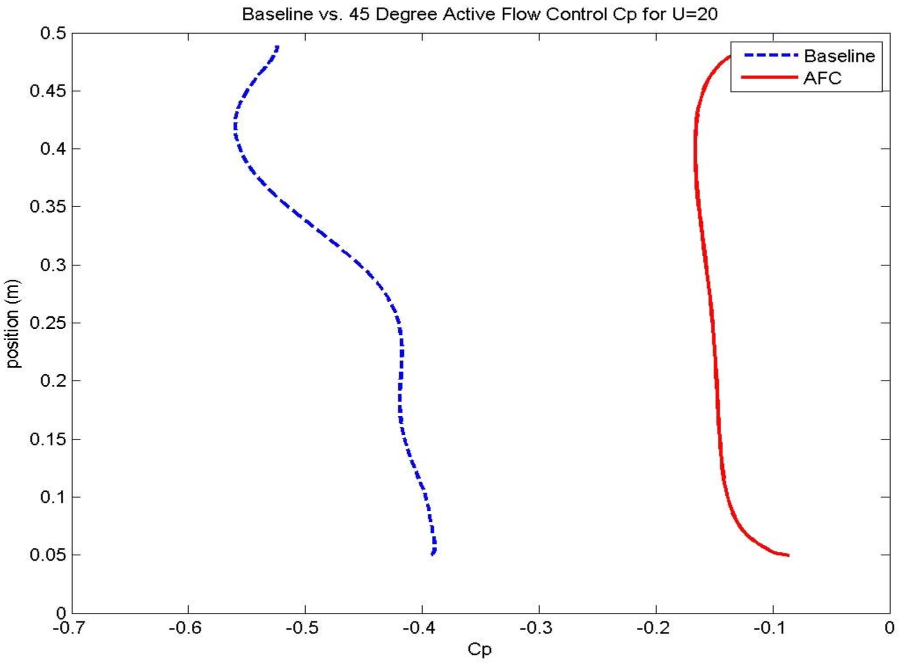


Figure 5.28 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at *Re* = 6.16 x 10^5

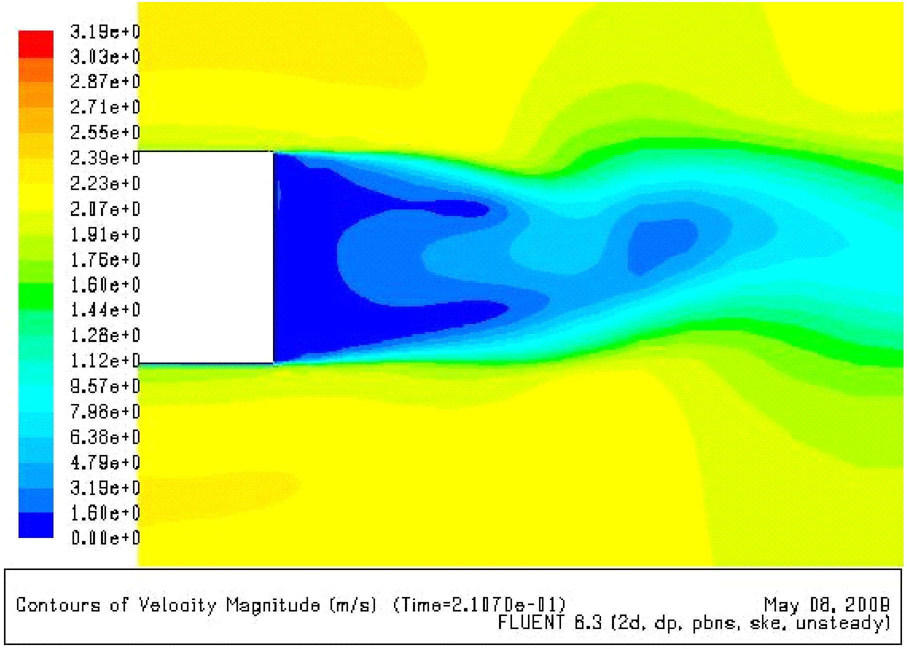


Figure 5.29 Computed velocity contours for flow with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at *Re* = 6.16 x 10^5

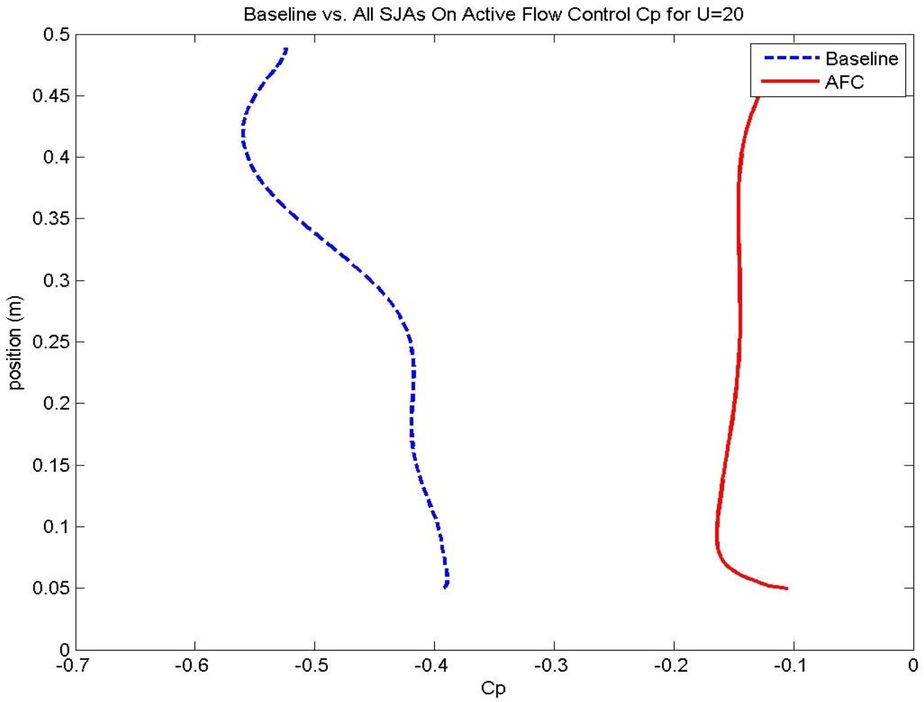


Figure 5.30 Baseline C_p versus C_p with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at *Re* = 6.16 x 10^5

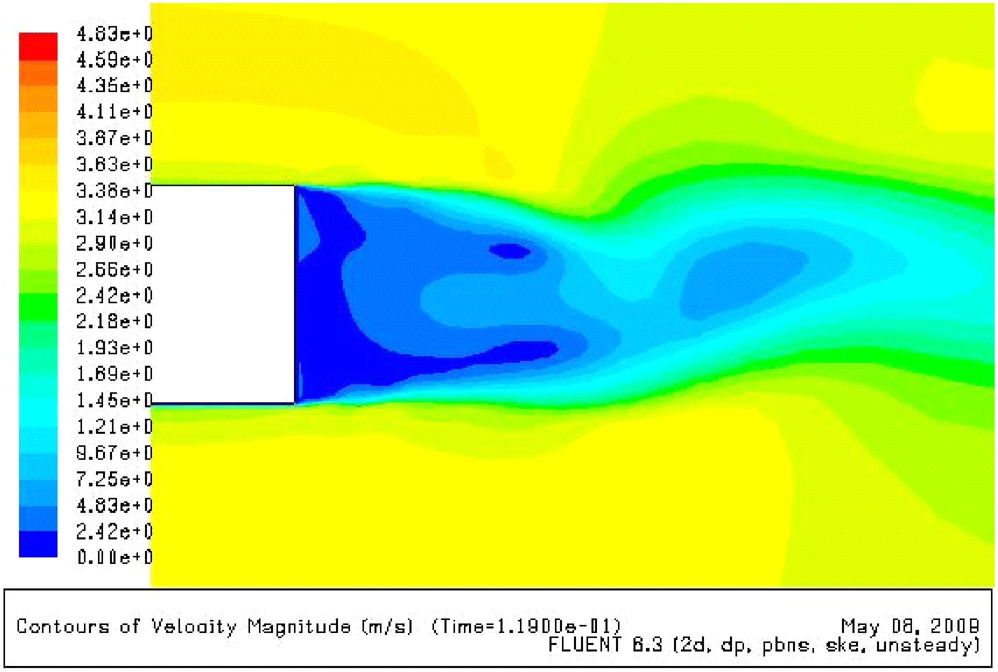


Figure 5.31 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at *Re* = 9.24 x 10^5

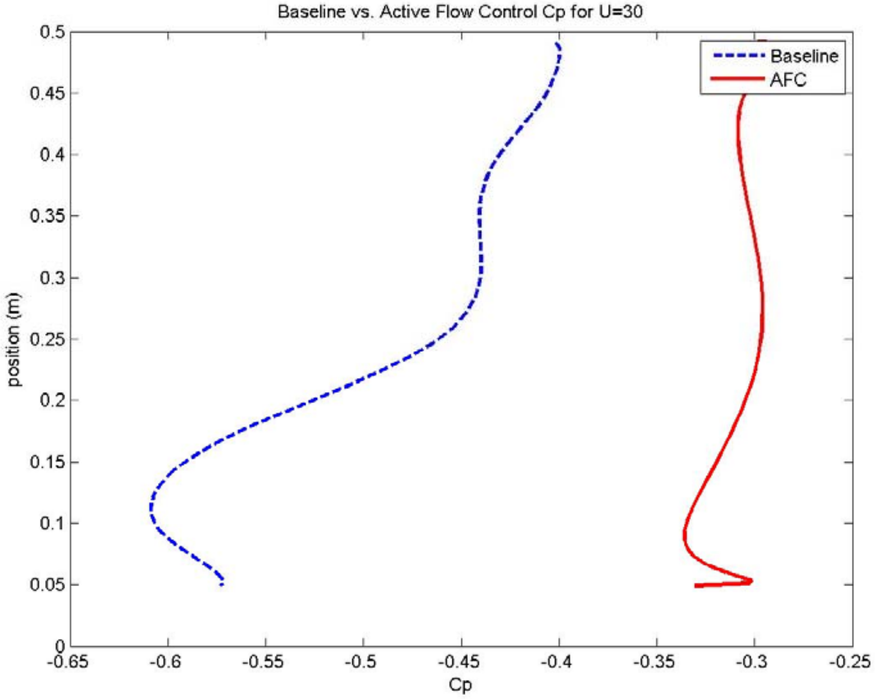


Figure 5.32 Baseline C_p versus C_p with active flow control for the generic truck body with two SJAs in configuration 1 (Figure 5.11) at $Re = 9.24 \times 10^5$

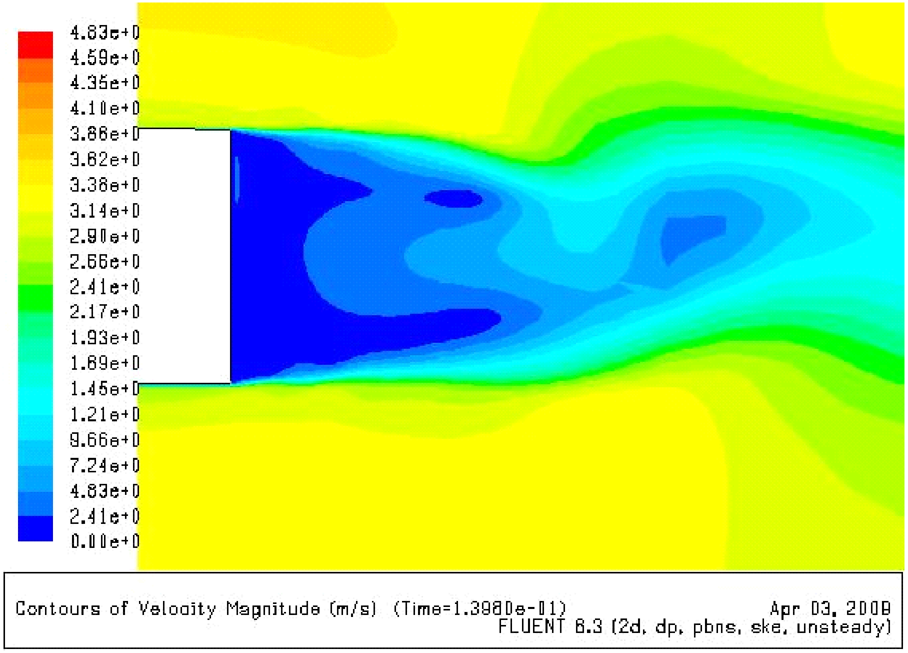


Figure 5.33 Computed velocity contours for flow with active flow control for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at $Re = 9.24 \times 10^5$

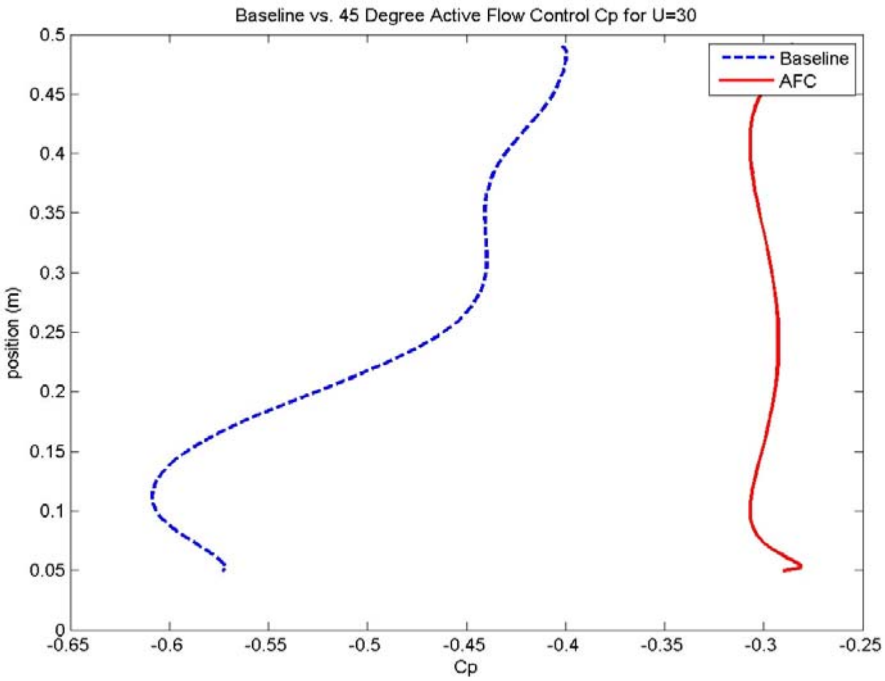


Figure 5.34 Baseline C_p versus C_p with active flow control C_p for the generic truck body with two SJAs in configuration 2 (Figure 5.12) at *Re* = 9.24 x 10^5

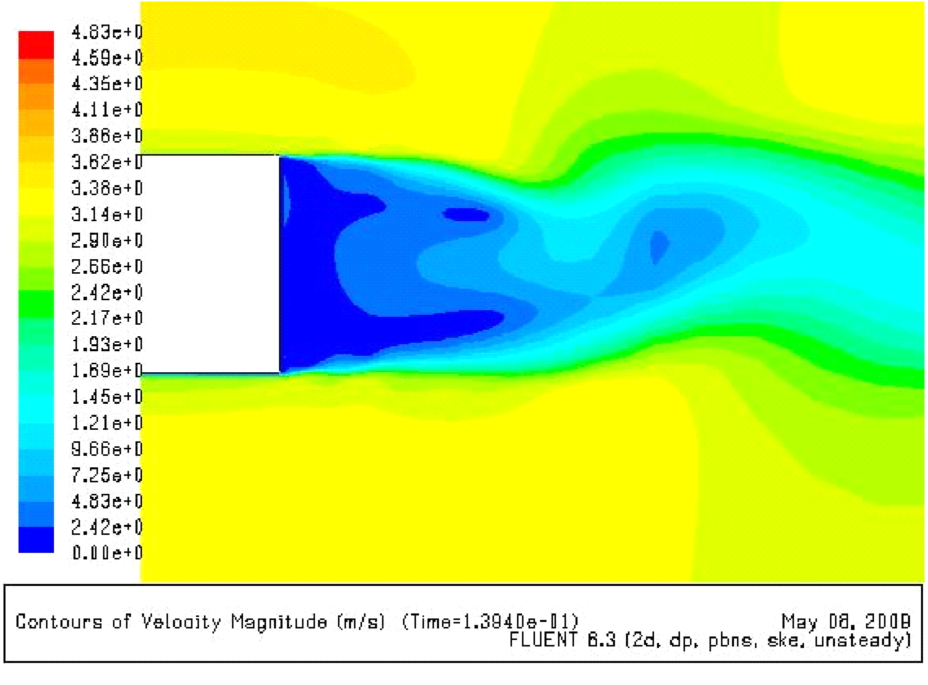


Figure 5.35 Computed velocity contours for flow with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at *Re* = 9.24 x 10^5

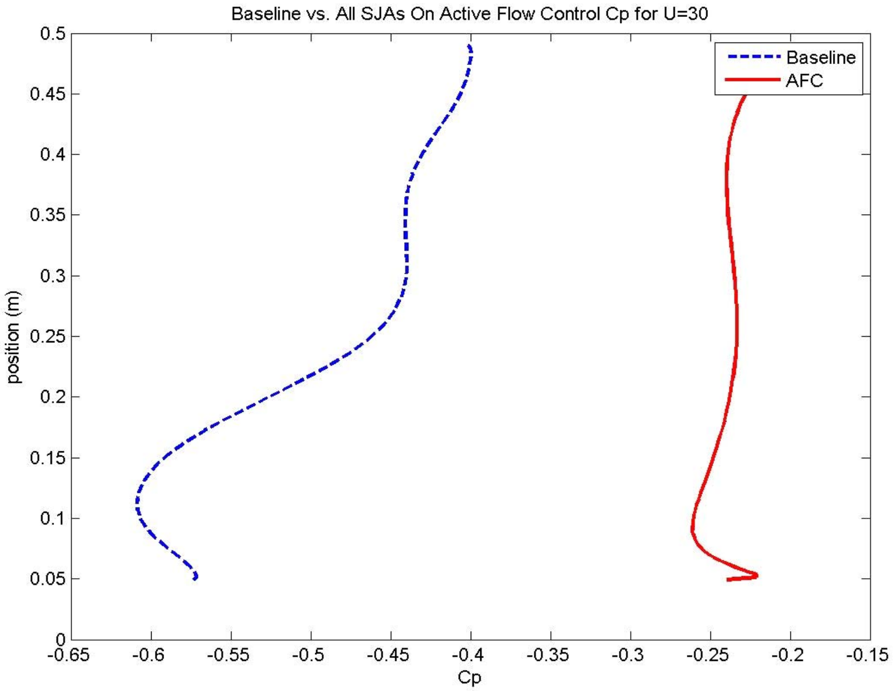


Figure 5.36 Baseline C_p versus C_p with active flow control for the generic truck body with three SJAs in configuration 3 (Figure 5.13) at $Re = 9.24 \times 10^5$

5.5 D-shaped Bluff Body

5.5.1 Introduction

Numerical simulations were performed on a two-dimensional D-shaped bluff body described in section 2.2. The computed baseline flow field was compared to the experimental data to determine an adaptive grid distribution and turbulence model which provided the closest agreement between the two. After the validation of the baseline case (without AFC), active flow control (AFC) was employed using two SJAs at an angle of 45° to the free stream direction (Figure 5.41) with parameters as in the experimental work of Pastoor et al. [7]. Detailed comparisons between the computations and experimental data were made for the flow field and the drag coefficient.

5.5.2 Computational Solution Procedure

Geometry Modeling and Grid Generation The D-shaped bluff body employed in the experiments of Pastoor et al. [7] is shown in Figure 2.3. In the experiment, the body was of the same width as the width of the wind tunnel which essentially eliminated the three-dimernsional effects. Therefore, we considered it reasonable to model the D-shaped bluff body of Figure 2.3 as a two-dimensional body for the numerical simulations. The body geometry (Figure 5.37) was created in GAMBIT with a complete computational domain and mesh shown in Figure 5.38. Figure 5.39 and Figure 5.40 show the zoomed-in view of the adaptive mesh near the body and the adapted mesh in the synthetic jet regions respectively. The mesh was created in a similar fashion as that for the generic truck body, described in section 5.4. It has a semi-circular farfield with a radius of 12.5L. The farfield behind the rear face is at a distance of 20L. Each farfield face is structured with rectangular cells. The mesh decreases in cell size as it moves from the farfield boundaries to the body surface. The mesh in the vicinity of the body, shown in Figure 5.39, is adapted for greater flow field resolution in regions where the drag-inducing structures are dominant. The resulting mesh has a node/cell/face count of 37,452/75,519/38,067. Two SJAs were placed on the back of the D-shaped bluff body at locations and angles with respect to the free stream direction as shown in Figure 5.41. The SJAs have slit widths of 1 mm, which is the same as in the experiment.

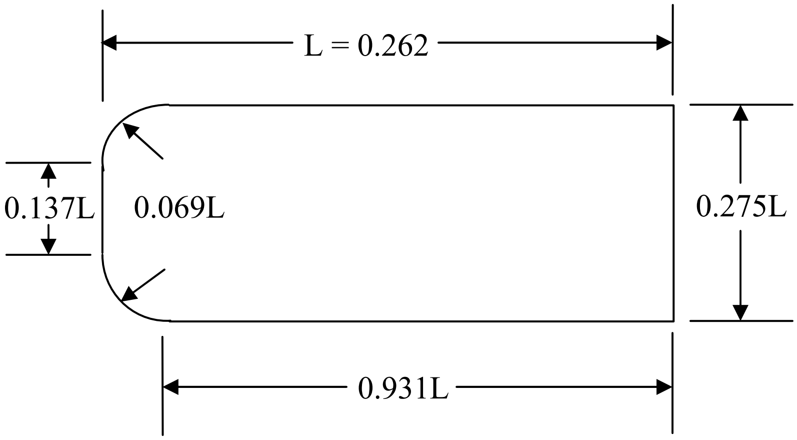


Figure 5.37 Geometry of the D-shaped bluff body [7]

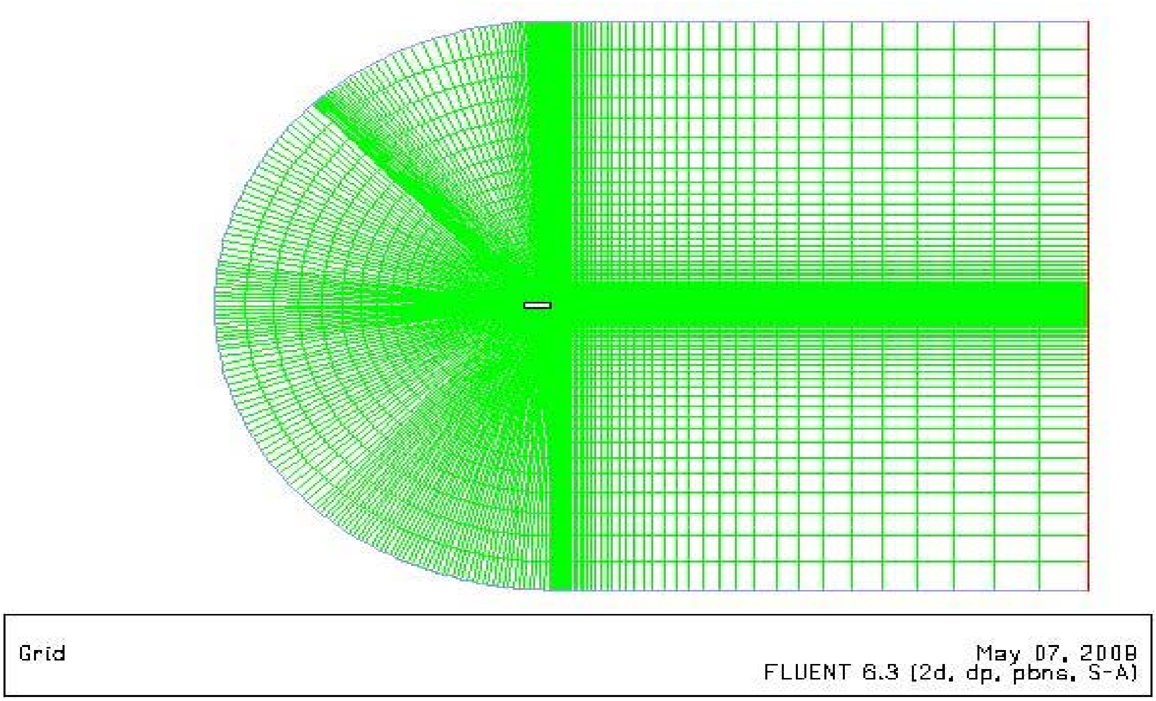


Figure 5.38 D-shaped bluff body: complete computational domain with structured mesh

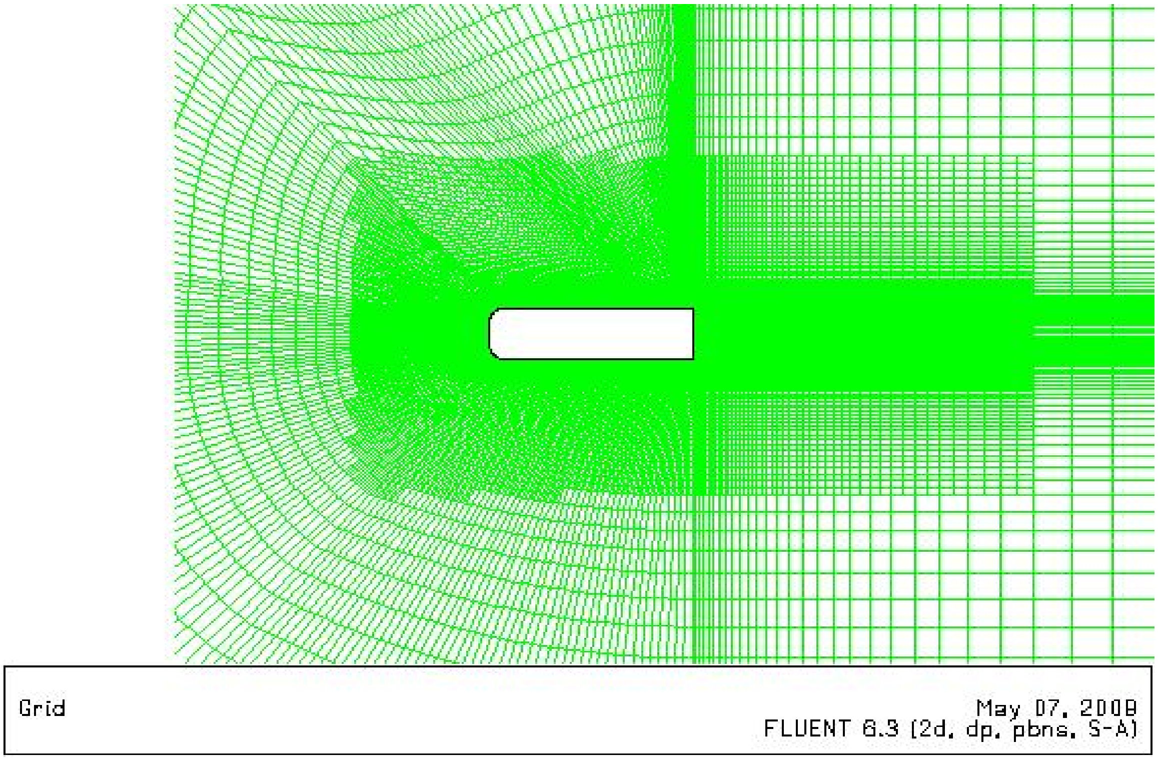


Figure 5.39 D-shaped bluff body with zoomed-in mesh near the body

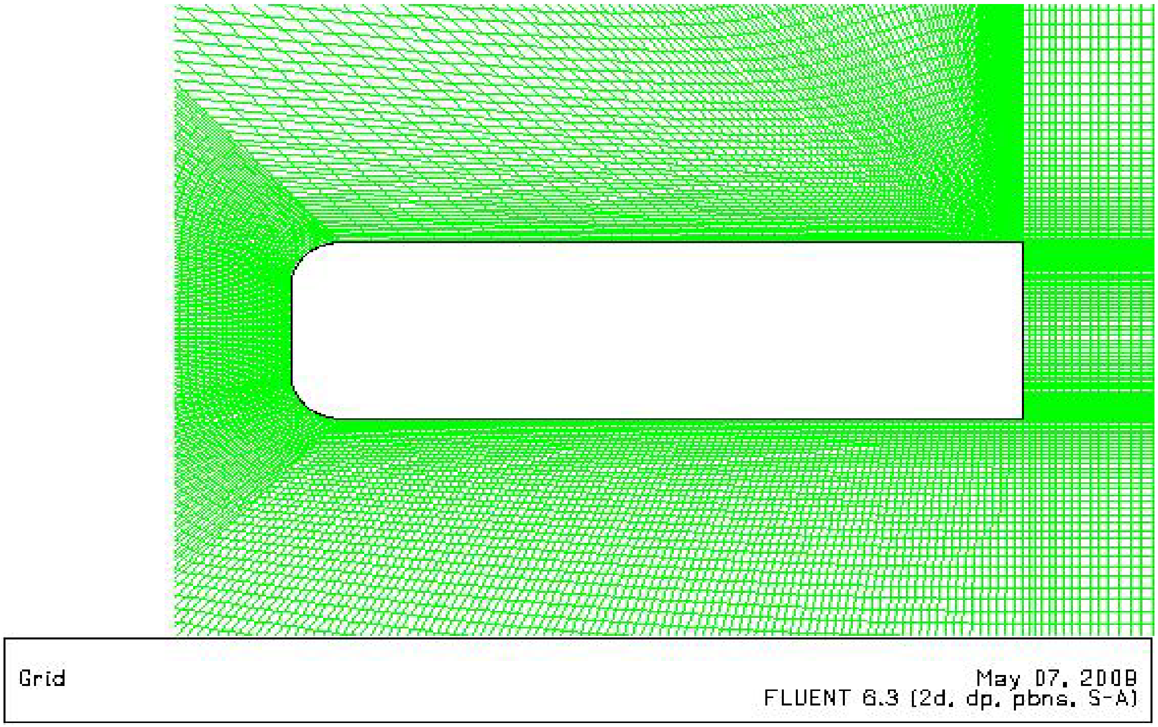


Figure 5.40 D-shaped bluff body with zoomed-in adapted mesh very close to the body and synthetic jet regions

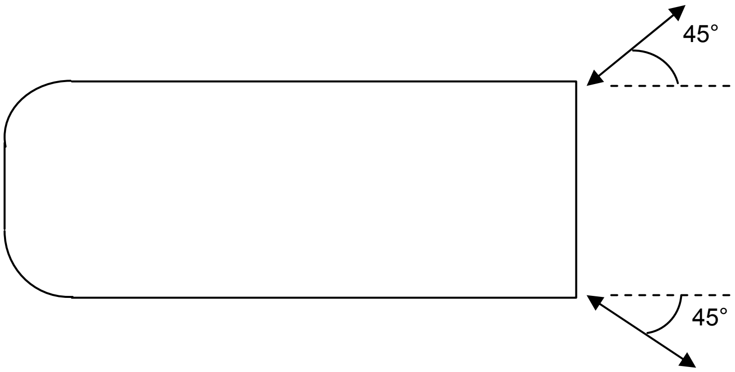


Figure 5.41 Two SJA locations and their exit angles on the rear face of the D-shaped bluff body

CFD Solution The commercial CFD solver FLUENT was employed to compute the flow field and the coefficient of drag. Baseline (without AFC) and active flow control flow fields were computed at Reynolds numbers of 35,000, 46,000, 58,000, and 70,000 based on the body height as the characteristic length. The free stream temperature and static pressure values were taken to be at sea level conditions; that is, 288.5 K and 101325 Pa respectively. The density of air ϱ = 1.225 kg/m^3 and the dynamic viscosity of air μ =1.849x10^{-5} kg/m-s were used. The Spalart-Allmaras (SA) turbulence model was employed in the calculations. The SJAs in Figure 5.41 were specified to have St = 0.15 and c_μ = 0.01. These were the optimal values of St and c_μ used in the experiments [7] to acheieve maximum drag reduction.

5.5.3 Results and Analysis

Experimental and numerical results for the D-shaped bluff body are given in Table 5.5 and Figure 5.42. Baseline (without AFC) numerical results are higher than the baseline experimental results by nearly 20%. However, both the computed and experimental results show an increase in the coefficient of drag with an increase in the Reynolds number. One of the reasons large discrepencies between the computed and experimental value of C_d could be that the numerical work was done in two dimensions while the experimental model was three-dimensional. Although the experimental body spanned from wall to wall in the wind tunnel, essentially eliminating any finite body effects, there may be significant wind tunnel wall effects that were not accounted for in the 2-D CFD model. In addition, the Reynolds numbers varied from 35,000 to 70,000 which are relatively low compared to the generic truck case of section 5.4. The SA turbulence model is not suitable at these Reynolds numbers. These two issues need further investigation.

Table 5.5 Experimental versus computed C_d for the D-shaped bluff body

Reynolds Number	**Experimental C_d (baseline case without AFC)**	**Experimental C_d with AFC**	**Computed C_d (baseline case without AFC)**	**Computed C_d with AFC**	**% Change in computed C_d with AFC**
35,000	0.92	0.77	1.14	0.98	13.8
46,000	0.89	0.75	1.16	0.96	17.1
58,000	0.90	0.76	1.18	0.95	19.7
70,000	0.91	0.76	1.20	0.95	21.3

Active flow control results also show a similar discrepency between the numerical and experimental coefficients of drag. This is expected because the two issues stated above also hold for flow with AFC. The experimental active flow control results show about 15 to 16% decrease in the coefficient of drag while the computed results show 14 to 21% decrease depending upon the Reynolds number as shown in Table 5.5 and Figure 5.42. Similar to the generic truck body (section 5.4), the numerical results for this case show larger decreases in the coefficient of drag with increasing Reynolds number.

Computed velocity contours for the baseline case (without AFC) are given in Figure 5.43 and Figure 5.44 for Reynolds numbers of 35,000 and 70,000, respectively. These plots show similar wake flow structures as for the generic truck body in section 5.4. Both figures exhibit the Kármán vortex street that characterizes a bluff body wake. Computed velocity contours with AFC are shown in Figure 5.45 and Figure 5.47. In this case, the SJAs have created much more symmetrical wake flow structures. However, unlike the generic truck body, the deadwater deadwater region is smaller, but is large enough to increase the base pressure and thus decrease the aerodynamic pressure drag of the body. Figures Figure 5.46 Figure 5.48 show the increase in computed base pressure with AFC compared to the base pressure without AFC.

Using Crowther and Gomes' power calculation, each SJA would require 0.0036%, 0.0080%, 0.016%, and 0.029% of the original propulsive power at Reynolds numbers of 35,000, 46,000, 58,000, and 70,000 respectively.

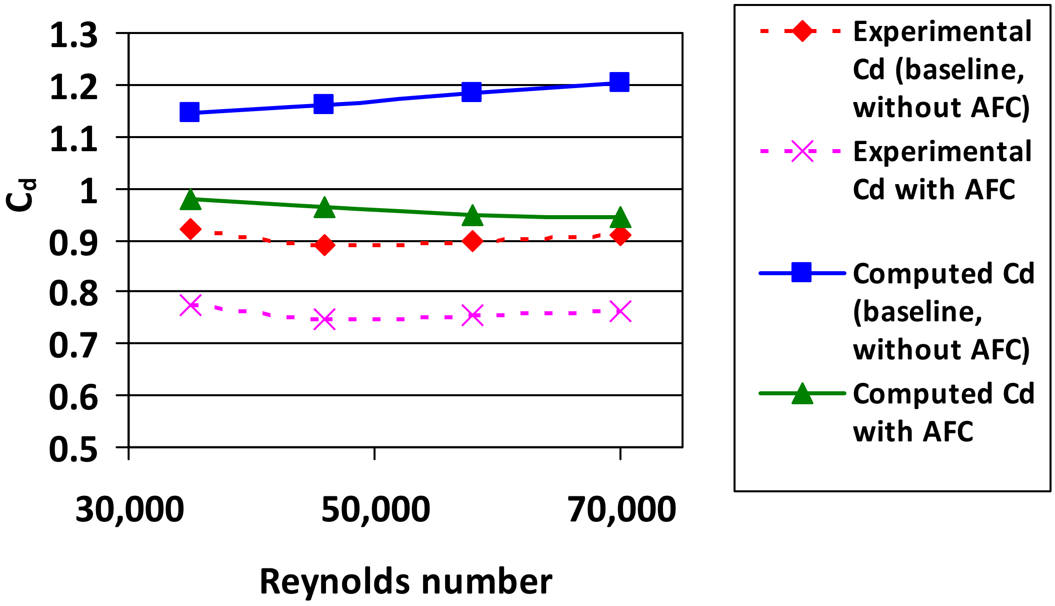


Figure 5.42 Experimental versus computed C_d for the D-shaped bluff body

Baseline

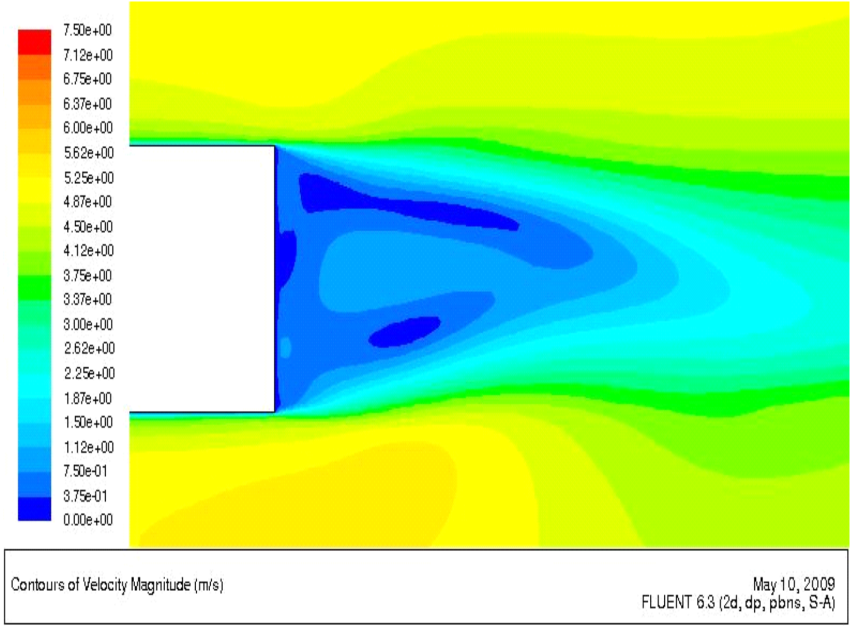


Figure 5.43 Computed velocity contours for the D-shaped bluff body for the baseline case (without AFC) at *Re* = 35,000

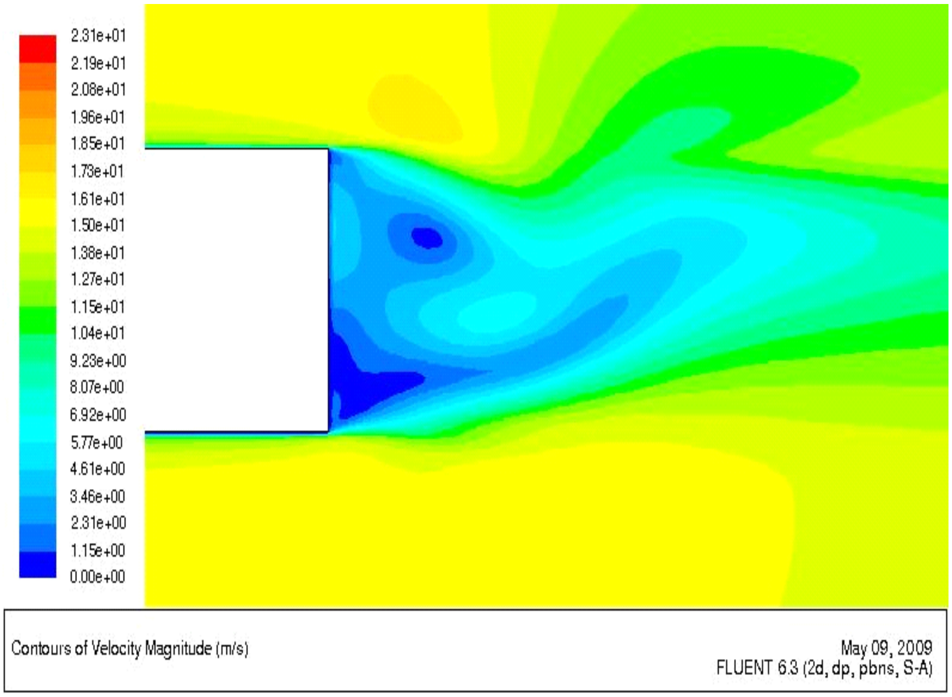


Figure 5.44 Computed velocity contours for the D-shaped bluff body for the baseline case (without AFC) at *Re* = 70,000

Active Flow Control

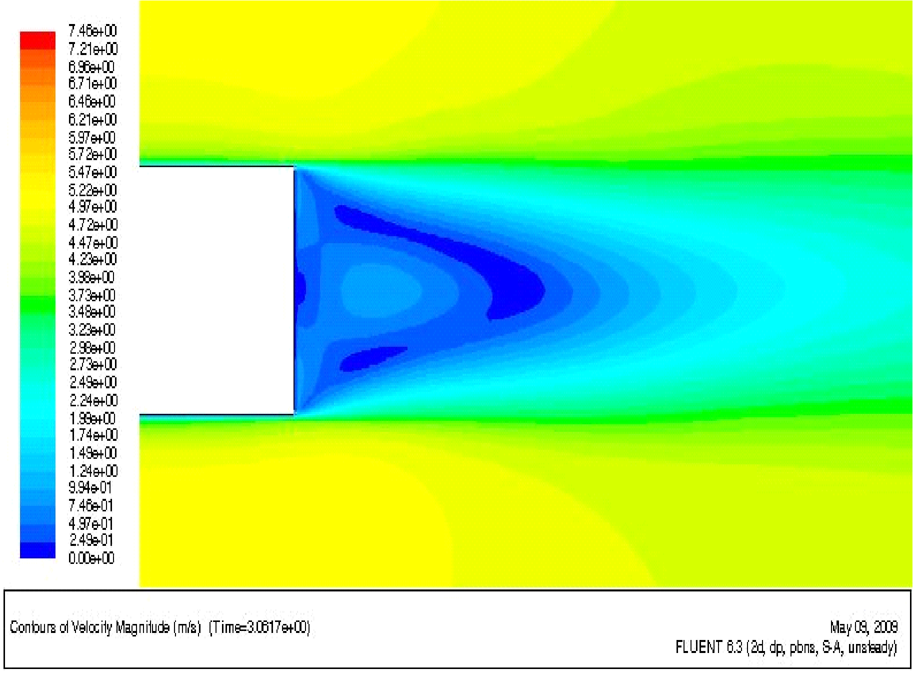


Figure 5.45 Computed velocity contours for the D-shaped bluff body with AFC at *Re* = 35,000

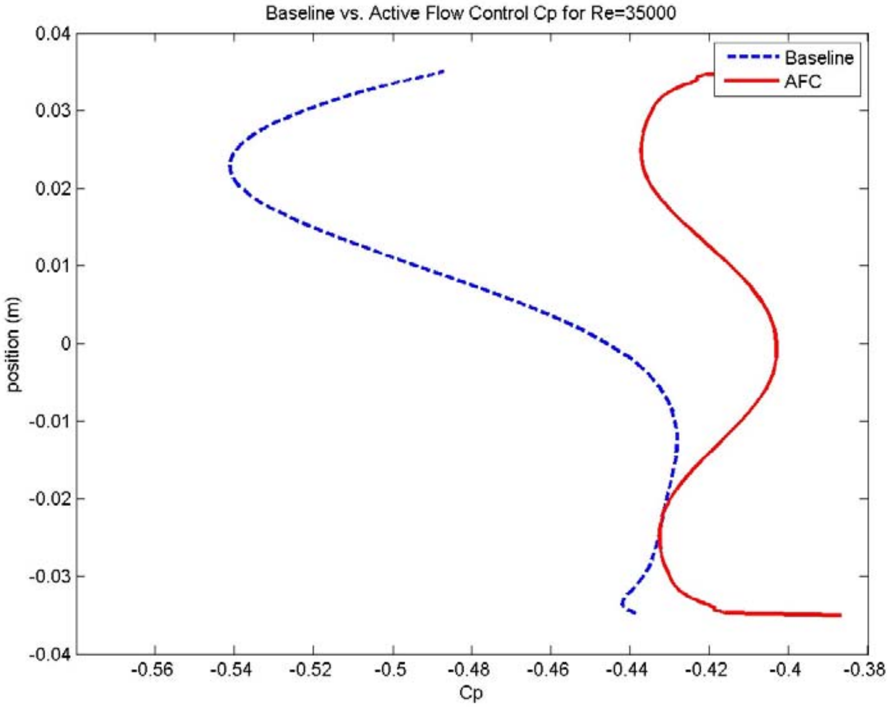


Figure 5.46 Computed C_p for the D-shaped bluff body for the baseline case (without AFC) and the case with AFC at *Re* = 35,000

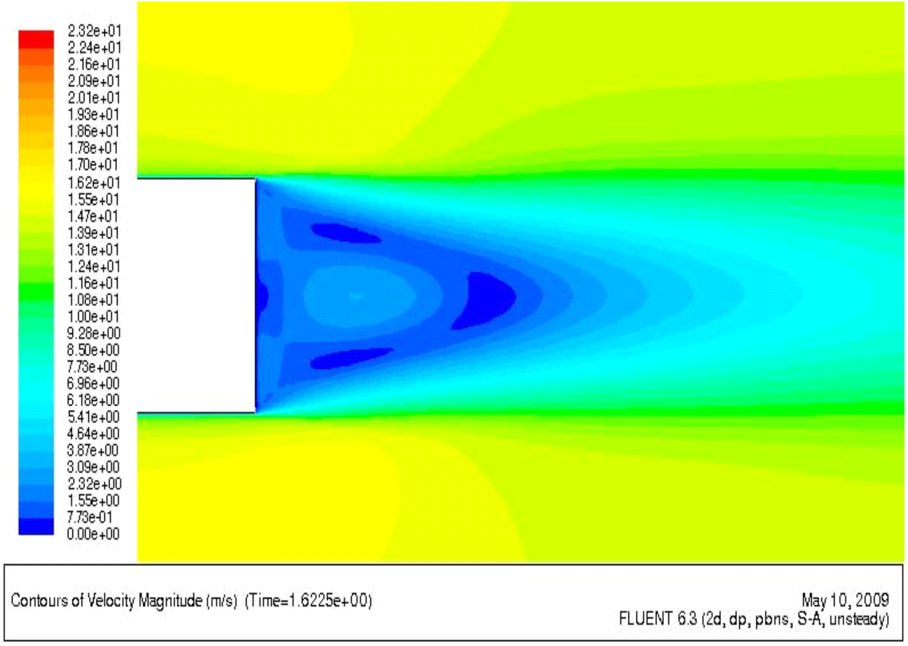


Figure 5.47 Computed velocity contours for the D-shaped bluff body with AFC at *Re* = 70,000

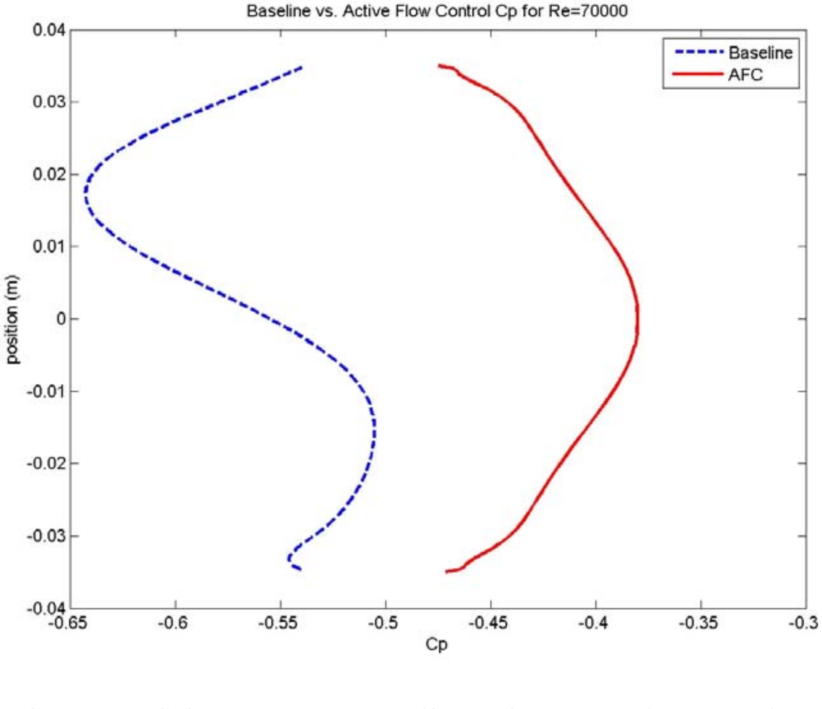


Figure 5.48 Computed C_p for the D-shaped bluff body for the baseline case (without AFC) and the case with AFC at *Re* = 70,000

5.6 Ahmed Body

5.6.1 Introduction

Numerical simulations were performed on the three-dimensional Ahmed body as given in sections 2.3 and 2.4. Computed baseline flow (without AFC) was compared to the experimental data to determine adaptive grid distribution and turbulence model which provided the closest agreement between the two. The grid was sufficiently dense so that the solution was almost grid independent. Active flow control was employed using simulated SJAs in two different configurations as given in the experimental work. Its usefulness was determined with final drag coefficient and analysis of wake structure and pressure at the truck base.

5.6.2 Computational Solution Procedure

Geometry Modeling and Grid Generation The Ahmed body geometry (Figure 5.49), from reference [18], was created in three dimensions in GAMBIT with a structured 3-D mesh shown in Figure 5.50. Figure 5.51 and Figure 5.52 show two different views of the mesh on the surface. Three dimensional meshing must be done carefully to ensure that all cell volumes mesh correctly with their surrounding cell volumes; they do not create negative cell volumes (or Jacobians) in the domain. Particular care was needed at the slant face and at SJA faces. The detailed dimensions of the body are shown in Figure 5.49; slant angle $\varphi = 25°$. In Figure 5.50, each meshed volume is structured with a rectangular parallelepiped cells. The mesh decreases in cell volume as it moves from the farfield to the truck so that the pertinent flow field structures are accurately resolved. The resulting mesh had a node/cell/face count of 271,248/829,131/286,775.

In one simulation, steady blowing actuators were placed on the back of the Ahmed body at locations shown in Figure 2.7. The slits were 2.1 mm in width. In another simulation, SJAs were placed on the back of the body at locations and angles shown in Figure 2.5. The slit widths of the SJAs were 5 mm.

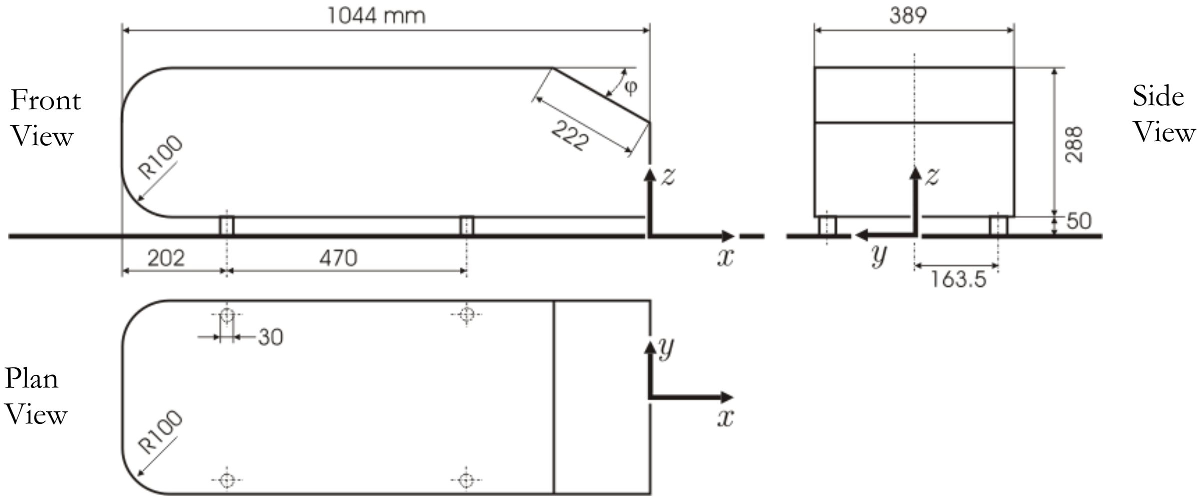


Figure 5.49 Ahmed body geometry (all dimensions are in mm)

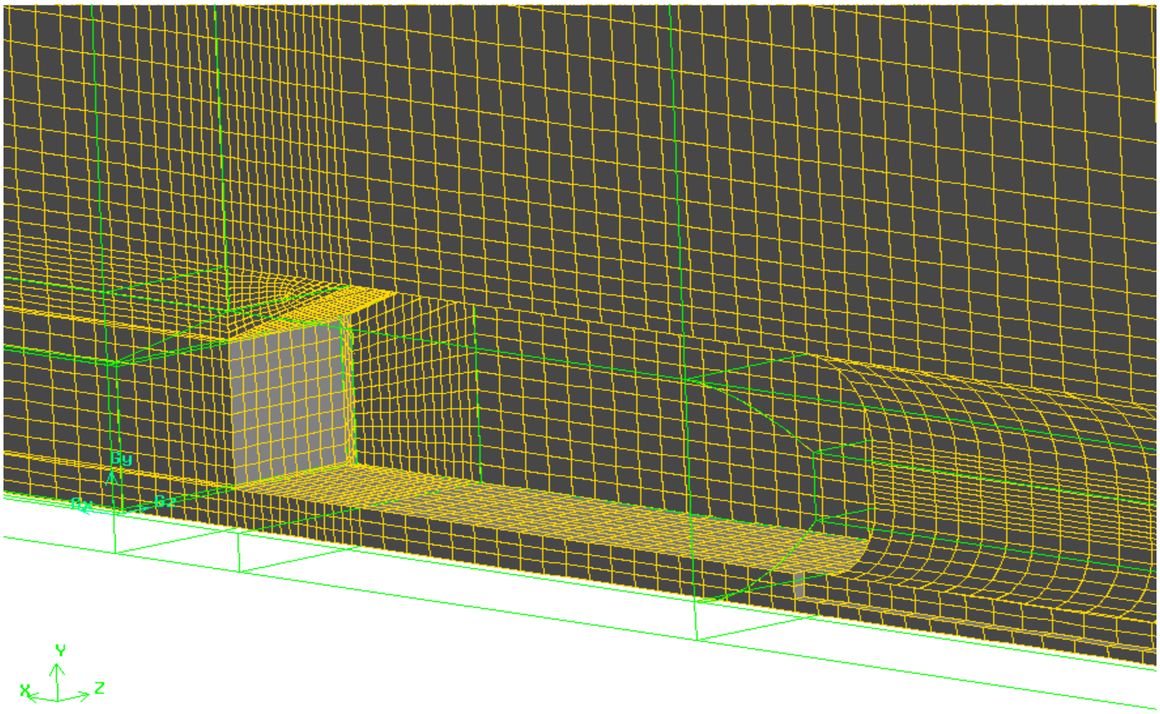


Figure 5.50 3-D structured mesh surrounding the Ahmed body

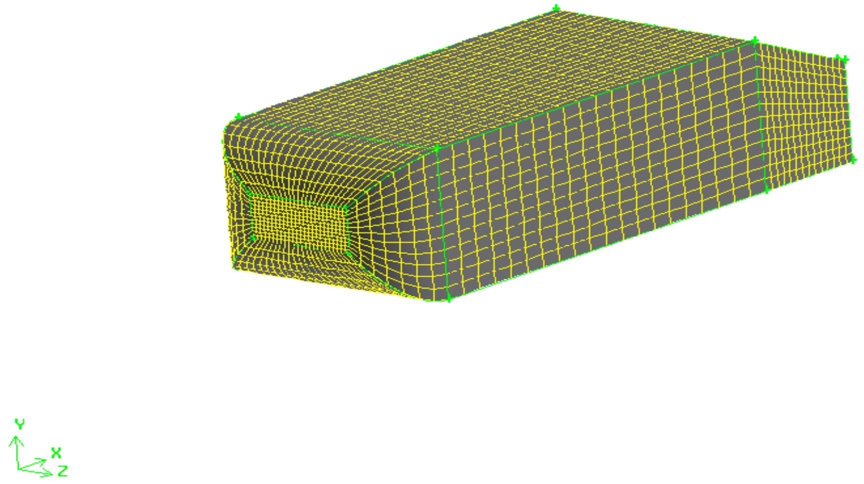


Figure 5.51 Surface mesh on the Ahmed body (view showing the details of the front part)

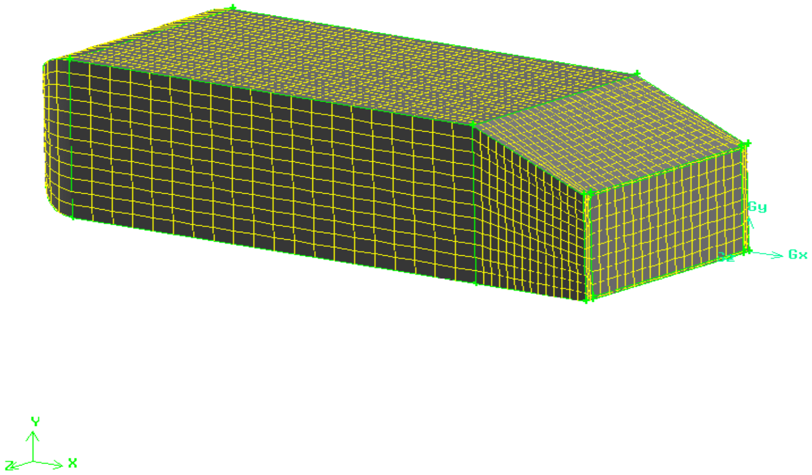


Figure 5.52 Surface mesh on the Ahmed body (view showing the details of the back part)

CFD Solution The commercial CFD solver FLUENT was employed to compute the flow fields and the coefficient of drag. Baseline (without AFC) and flow fields with active flow control were computed at a Reynolds numbers of 500,000 with the body height as the characteristic length. Free stream temperature and static pressure values were taken to be at sea level conditions; 288.5 K and 101325 Pa respectively. The density of air ϱ = 1.225 kg/m^3 and the dynamic viscosity of air μ =1.849x10^{-5} kg/m-s were used. The Spalart-Allmaras (SA) turbulence model was employed in the calculations. In the simulation with steady blowing, the steady blowing jets had blowing velocity equal to the free stream velocity. In the simulation with SJAs, the SJAs were specified with $St = 0.43$ and $c_\mu = 0.009$. These values were taken from reference [8].

5.6.3 Results and Analysis

The Ahmed body geometry was initially created in two-dimensions for reasons of simplicity to allow for less computational time. However, the baseline computed coefficient of drag was found to be lower than the experimental value. This result led to a closer examination of the wake flow structures found in the experimental work. It was found that while spanwise two-dimensional vortex structures occured at the back edges of the Ahmed body similar to the wake structures generated by the 2-D generic truck body and D-shaped bluff body, the Ahmed body also exhibited longitudinal vortex structures (because of an aspect ratio of cross-section close to unity) that could only be resolved in three dimensions. These vortices were generated along the slant edges and the vortex plane was perpendicular to the free stream. Thus, these vortices would have never been realized in a two-dimensional simulation.

The three-dimensional simulation for the baseline case (without AFC), while computationally more expensive, gave the numerical value of the coefficient of drag within 1% of the experimental result as shown in Table 5.6. Figure 5.53 shows the spanwise vortices behind the Ahmed body. The longitudinal vortex structures are shown in Figure 5.54. Pressure contour plots for baseline flow are shown in Figure 5.55. Figure 5.56 shows the computed pressure contours on the front face of the Ahmed body for the baseline flow at Reynolds number of 500,000. Figure 5.57 shows regions of low pressure along the slant edges where the longitudinal vortices are produced. This figure also shows a region of low pressure at the top of the base where the flow initially separates.

Table 5.6 Experimental versus computed drag coefficient C_d for the Ahmed body

Case	**Reynolds Number**	**Experimental C_d**	**Computed C_d**
Baseline	500,000	0.374	0.378
Steady Blowing	500,000	0.350	0.385

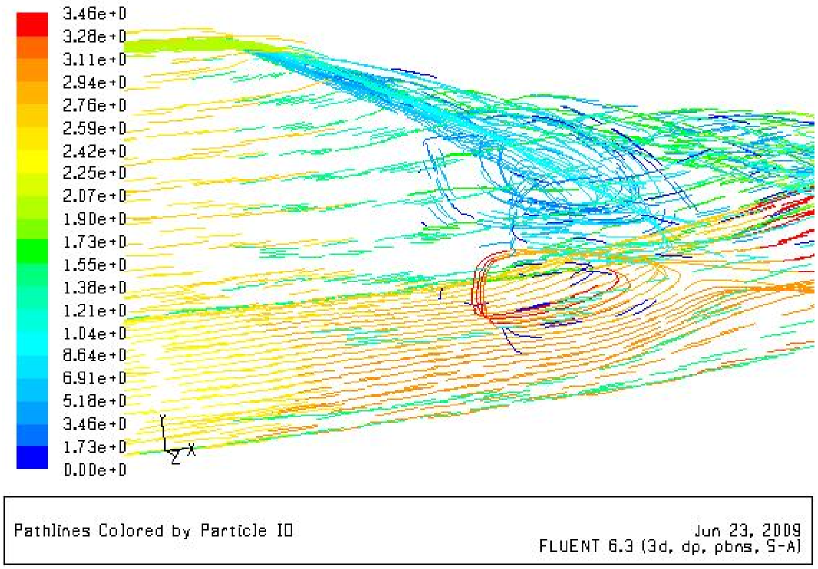


Figure 5.53 Spanwise vortices behind the Ahmed body

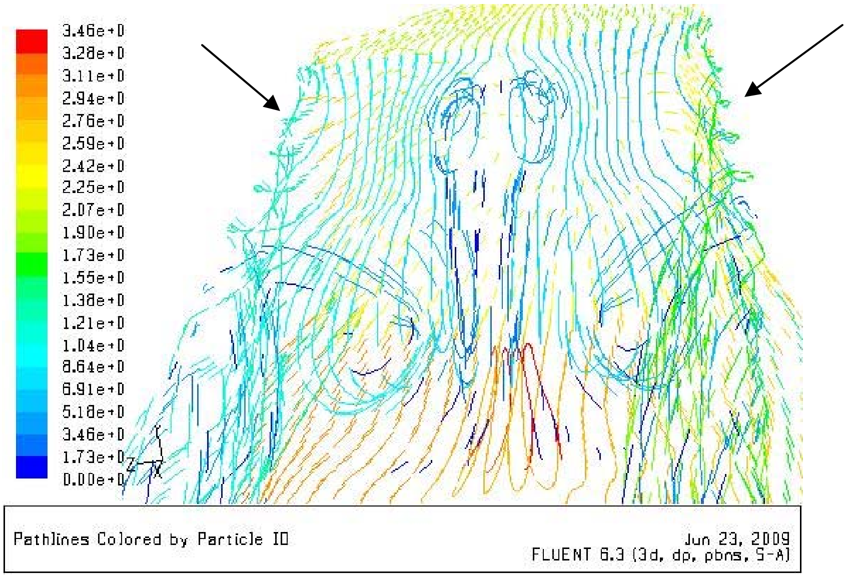


Figure 5.54 Longitudinal vertical structures behind the Ahmed body

Active flow control was applied using steady blowing from the four edges that define the base of the Ahmed body. The AFC with steady blowing was implemented to compare the results with the experimental data [10]. As shown in Table 5.6, numerical results show that the steady blowing has a negative effect on the coefficient of drag; the coefficient of drag increases. Experiments however show a small decrease in drag coefficient C_d. Additional numerical simulations with finer grids and alternative turbulence models need to be explored to improve the numerical prediction and make it closer to the experimental value of the drag coefficient. Figure 5.58 and Figure 5.59 show a noticeable pressure decrease in the base region. Both the slanted back and vertical back regions show an increase in the size of the area of the low pressure region as well as a decrease in the pressure magnitude. The experimental results showed a 6.4% decrease in the coefficient of drag with steady blowing; however, the overall efficiency using AFC increased by only 0.8% because the SJAs used 5.6% of the original propulsive power. With such a small increase in efficiency using AFC, steady blowing does not appear to be a strong candidate for achieving drag reduction.

Baseline

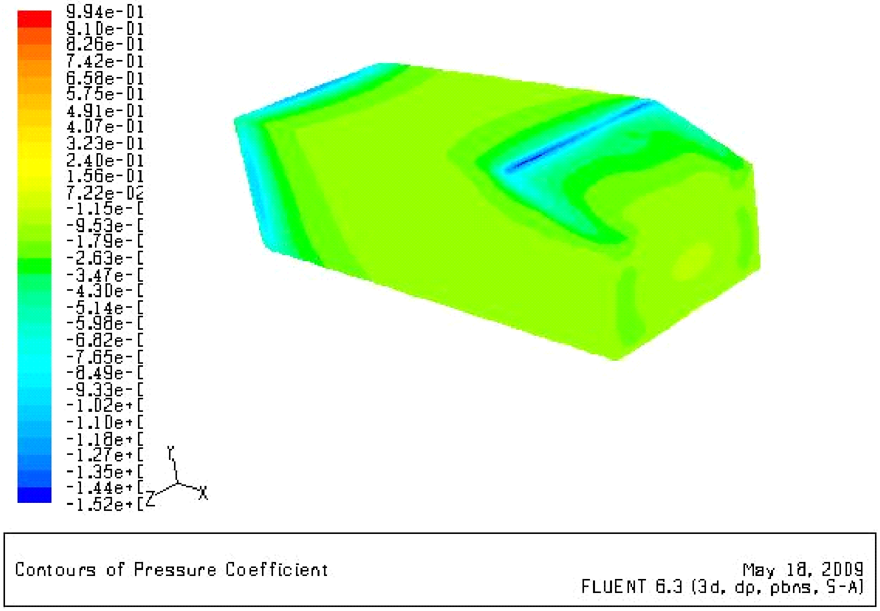


Figure 5.55 Isometric view of computed C_p contours for baseline flow (without AFC) past the Ahmed body, *Re* = 500,000

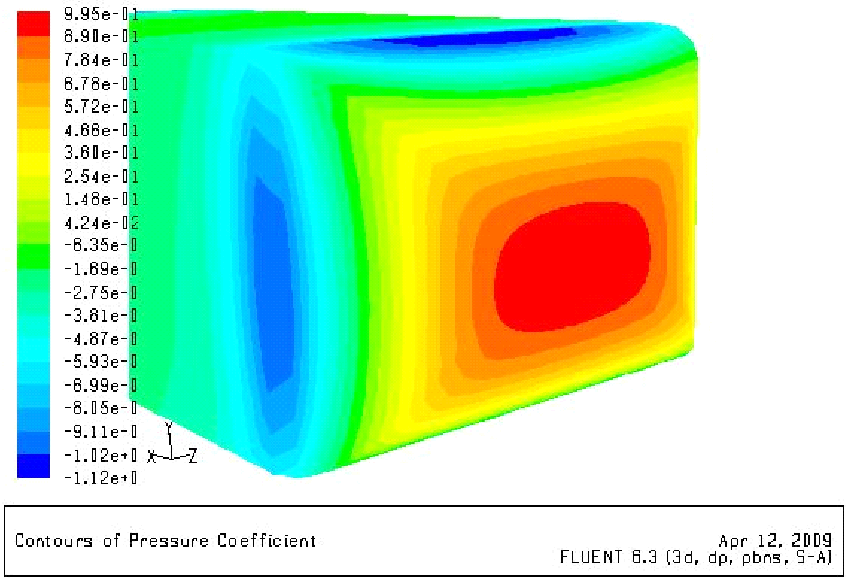


Figure 5.56 Computed C_p contours on the front face of the Ahmed body for baseline flow (without AFC), *Re* = 500,000

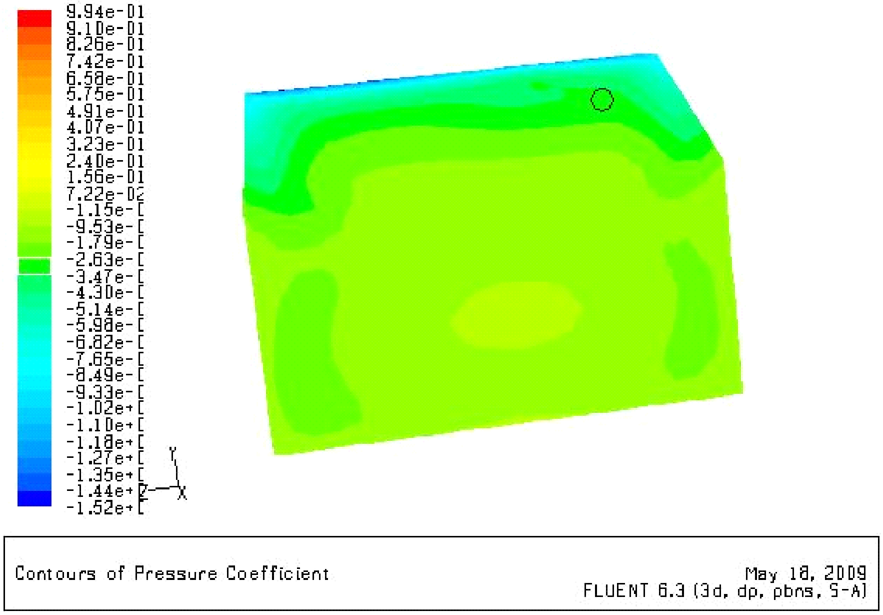


Figure 5.57 Computed C_p contours on the rear face of the Ahmed body for the baseline flow (without AFC), *Re* = 500,000

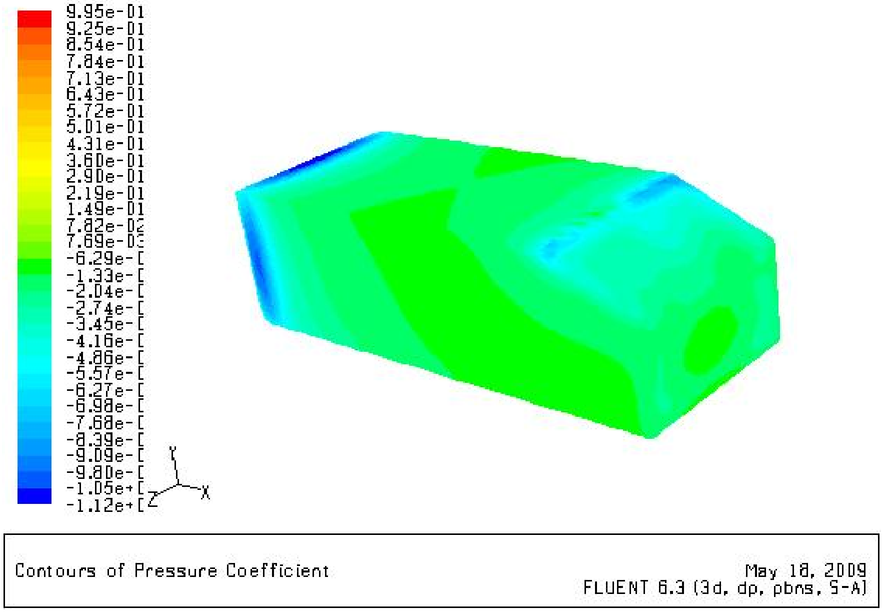


Figure 5.58 Isometric view of computed C_p contours on the Ahmed body for AFC with steady blowing, *Re* = 500,000

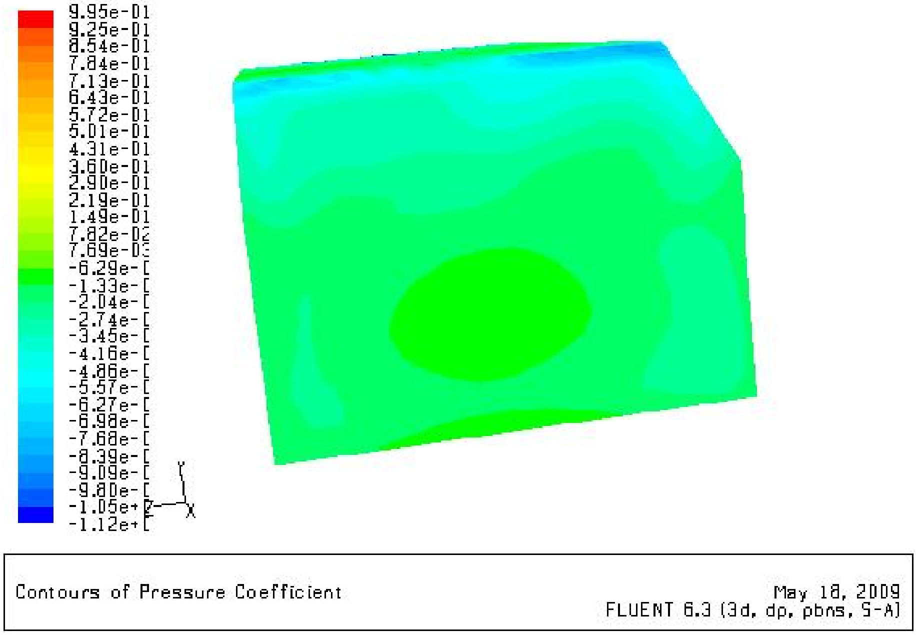


Figure 5.59 Computed C_p contours on the rear face of the Ahmed body for AFC with steady blowing, Re = 500,000

5.7 Conclusions

In this chapter we studied the flow fields of three different generic truck bodies by numerical simulations. The three configurations chosen in this study were used because of the availability of experimental data. Numerical computations were performed for baseline cases (without AFC), and for flow fields with AFC using Synthetic Jet Actuators (SJAs). Computations showed reasonable agreement with the experimental data for the coefficient of drag for most of the cases considered. Both the computations and experiments show that it is possible to achieve significant drag reduction by active flow control.

The first case, the generic truck body of Seifert et al. [5], was modeled in two dimensions. For this case, the baseline numerical drag coefficients agreed within 5% of the experimental values at Reynolds numbers ranging from 3.08 x 10^5 to 9.24 x 10^5. However, the numerical results showed an expected decrease in the drag coefficient with increasing Reynolds number while the experiments showed an increase in the drag coefficient with increasing Reynolds number. This discrepancy is currently being resolved in consultation with the author of the experiment as well as by conducting additional numerical simulations. Active flow control, using SJAs in three different configurations on the rear face of the body, showed 9 to 16% reduction in drag coefficient depending upon the Reynolds number and the parameters of the SJAs (frequency, amplitude, width, and location).

The second case of the D-shaped bluff body of Pastoor et al. [7] was also modeled in two dimensions. For this case, the baseline (without AFC) numerical drag coefficients agreed within 20% of the experimental values. The source of this large discrepancy between the computed and experimental value of C_d perhaps comes from the fact that the computations were performed in two dimensions while the experimental work was done in three dimensions. The other sources of error in numerical results, namely the grid resolution and turbulence model, are unlikely to be the cause of such a large discrepancy. Using active flow control, with two SJAs placed at the top and bottom corners of the rear face of the body and inclined at 45° to the free stream (Figure 5.12), numerical simulations showed a 14 to 21% reduction in drag depending upon the Reynolds number compared to the baseline (without AFC) case. Experimental results found similar reductions in drag.

The third case of the Ahmed body [8, 10] was modeled in three dimensions. For this case, the baseline (without AFC) numerical drag coefficients agreed within 1% of the experimental value at a Reynolds number of 500,000. While this simulation was computationally intensive, it validated the numerical methodology—grid resolution, turbulence model, numerical solver, etc. Using active flow control, with steady blowing at the back of the body, numerical simulations found an increase in drag. This was contrary to the experimental result which showed a 6.4% decrease in drag coefficient. This discrepancy requires further investigation.

It should be noted that a 15% reduction in aerodynamic pressure drag of the vehicle is equivalent to about 5% in fuel savings. Numerical simulations show promise of significant aerodynamic pressure drag reduction of truck-shaped vehicles by manipulating the flow in the wake region using AFC. Using AFC on millions of ground vehicles has the potential of saving millions of gallons of fuel and of reducing operating costs as well as greenhouse gas emmisions.

Chapter 6

Future Work

The results presented in this thesis clearly demonstrate the potential of both shape optimization and active flow control techniques for reducing the aerodynamic drag of ground vehicles. The results also point the way in several directions for future work.

For shape optimization, additional studies should be conducted by employing different constraints for the control and anchor points. It would be interesting to let the genetic algorithm allow free movement of both anchor points in the x-coordinate direction or to remove the restraint for the control points in the y-coordinate direction. Finally, the genetic algorithm should be extended to three dimensions for shape optimizations of 3-D truck fronts.

In applying the active flow control with Synthetic Jet Actuators (SJAs), the influence of various parameters of SJAs on the effectiveness of flow control should be studied. The SJA parameters include velocity amplitude, frequency, slit width, SJA location, SJA angle, phase among SJAs, and number of SJAs. Previous experiments [7, 8] have used feedback closed-loop control models for finding the best velocity amplitude and frequency parameters. Perhaps, a genetic algorithm could be developed and employed to find the best combination of SJA parameters for achieving optimal effectiveness using AFC.

While this thesis has demonstrated the effectiveness of SJAs on drag reduction for several bluff bodies, suction and oscillatory blowing (SaOB) actuation devices have also been proven useful [6] for drag reduction on bluff bodies. These devices can also be easily attached to the back of a truck. Additional simulation on SaOB devices should be conducted to evaluate their effectiveness for drag reduction vis-à-vis SJAs.

To improve the numerical simulation results for the D-shaped bluff body, three-dimensional numerical computations should be performed with and without active flow control. Similarly, additional three-dimensional computations for the Ahmed body should be performed with active flow control.

All the future work suggested above assumes the generic truck shape bodies are surrounded only by the free stream, without the presence of the ground. The ground effect must be included for realistic determination of drag without and with AFC, both in experimental and numerical simulations.

Finally, the shape optimization and active flow control methods should be combined to find the largest amount of drag reduction possible. Based on the results presented in this thesis, total drag reduction of about 25% (compared to the vehicle drag without shape optimization and AFC) appears to be feasible.

Appendix A

Java Code for Airfoil Shape Optimization Using a Genetic Algorithm

```
/**
 * GALite.java
 * Author: Brandon Morgan
 *
 * This is the main class used in optimization.  It represents the
optimization
 * code from GAANNFluent separated from the GUI components.
 */

package galite;

import ann.ANNAirfoil;
import ann.OnlineANN;
import ga.Generation;
import ga.Individual;

public class GALite {
      public static final int DEBUG_LEVEL = 2;
      public static OnlineANN net = null;

      private double mutRate, removePercentage;
      private int genSize, numGens, E;
      private Airfoil bestIndividual;
      private Generation gen;

      public GALite(double mutRate,double removePercentage,int
genSize, int numGens){
            this.mutRate = mutRate;
            this.removePercentage = removePercentage;
            this.genSize = genSize;
            this.numGens = numGens;
            E = (int) Math.round(genSize * this.removePercentage);


            //Set up the ANN
            try{
                  net = new OnlineANN(20);
                  ANNAirfoil foil1 = new ANNAirfoil(-
0.35,0.0,36.70146039826009);
                  //foil1.setObjective(12.645768628); // Cl/Cd
                  foil1.setObjective(0.40683743); //Cl
                  ANNAirfoil foil2 = new ANNAirfoil(-
2.0,10.0,44.118199448833565);
                  //foil2.setObjective(33.322463795); // Cl/Cd
                  foil2.setObjective(1.744573); // Cl
                  ANNAirfoil foil3 = new ANNAirfoil(-
0.35,0.0,0.4436541384011625);
                  //foil3.setObjective(0.138968970); //Cl/Cd
                  foil3.setObjective(0.033142283); //Cl
                  ANNAirfoil foil4 = new ANNAirfoil(-
0.35,10.0,13.095334310034984);
                  //foil4.setObjective(7.295466139); //Cl/Cd
                  foil4.setObjective(2.1870077); //Cl
                  for(int i=0;i<8;i++){
                        net.addTrainingPt(foil1);
                        net.addTrainingPt(foil2);
                        net.addTrainingPt(foil3);
                        net.addTrainingPt(foil4);
                  }
            }
            catch(Exception e){
                  e.printStackTrace();
                  System.exit(-1);
            }
      }

      public Airfoil runOptimization(){
            gen = new Generation(genSize);
      while(!gen.isFull()){
            gen.addIndividual(generateIndividual());
      }

      bestIndividual = (Airfoil) gen.getBestIndividual();

      if(DEBUG_LEVEL > 1)
            System.out.println("***** Generation 0  *****");
      if(DEBUG_LEVEL > 1)
            System.out.println(gen);
      if(DEBUG_LEVEL > 0)
            System.out.println("(0) Best Ind: " +
bestIndividual.getFitness());

      for(int i=1; i<=numGens; i++){
            advanceGen();
            if(DEBUG_LEVEL > 1)
                  System.out.println("***** Generation " + i + "
*****");
            if(DEBUG_LEVEL > 1)
                  System.out.println(gen);
            if(DEBUG_LEVEL > 0)
                  System.out.println("(" + i + ") Best Ind: " +
bestIndividual.getFitness());
      }

            return bestIndividual;
      }
```

```
	private Airfoil generateIndividual(){
		double a = Math.random() * AirfoilModifier.minA;
	double b = Math.random() * AirfoilModifier.maxB;
	double t = Math.random() * AirfoilModifier.maxThck;
	double m = Math.sqrt(b*b + a*a);
	double delta = Math.PI/2.0 + Math.atan(-a/b);
	double r = Math.sqrt( Math.pow(-1.0/(0.77*t) *
m*Math.cos(delta) - a,2.0) + Math.pow(b,2.0) );

	Airfoil af = new Airfoil();
	AirfoilModifier.modAirfoil(af, a, b, r);
		return af;
	}

	private void advanceGen(){
		Generation nextGen = new Generation(genSize);

		//Multiplication
		int count =0;
		while(count<E){
			Airfoil i1 = (Airfoil)
gen.getIndividual((int)Math.floor(Math.random()*genSize));
			Airfoil i2 = (Airfoil)
gen.getIndividual((int)Math.floor(Math.random()*genSize));
			if(i1!=i2 ){
				nextGen.addIndividual(crossover(i1,i2));
				count++;
			}

		}
		naturalSelection();
		Individual[] survivors = gen.getIndividuals();
		for(int i=0;i<survivors.length;i++){
			nextGen.addIndividual(survivors[i]);
		}

		mutate(nextGen);

		gen = nextGen;


		if(bestIndividual == null ||
gen.getBestIndividual().getFitness() > bestIndividual.getFitness()){
			bestIndividual = new
Airfoil((Airfoil)gen.getBestIndividual());
			if(DEBUG_LEVEL > 1)
				System.out.println("new best individual: " +
bestIndividual.getFitness());
		}
	}

	private Airfoil crossover(Airfoil i1, Airfoil i2){
		double a1,a2,b1,b2,r1,r2;

		if(i1.getFitness() > i2.getFitness()){
```

```
                  a2 = i1.A;
                  b2 = i1.B;
                  r2 = i1.R;
                  a1 = i2.A;
                  b1 = i2.B;
                  r1 = i2.R;
            }
            else{
                  a2 = i2.A;
                  b2 = i2.B;
                  r2 = i2.R;
                  a1 = i1.A;
                  b1 = i1.B;
                  r1 = i1.R;
            }

            double a = Math.random()*(a2-a1) + a2;
            double b = Math.random()*(b2-b1) + b2;
            double r = Math.abs(Math.random()*(r2-r1) + r2);

            Airfoil af = new Airfoil();
            AirfoilModifier.modAirfoil(af, a, b, r);

            return af;
      }

      private void naturalSelection(){
            gen.removeIndividuals(genSize-E);
      }

      private void mutate(Generation nextGen){
            for(int i=0;i<nextGen.getTotalNum();i++){
                  if(Math.random() <= mutRate){
                        Airfoil ind = (Airfoil)
nextGen.getIndividual(i);

                        double a = Math.random() *
AirfoilModifier.minA;
                  double b = Math.random() * AirfoilModifier.maxB;
                  double t = Math.random() *
AirfoilModifier.maxThck;
                  double m = Math.sqrt(b*b + a*a);
                  double delta = Math.PI/2.0 + Math.atan(-a/b);
                  double r = Math.sqrt( Math.pow(-1.0/(0.77*t) *
m*Math.cos(delta) - a,2.0) + Math.pow(b,2.0) );

                        Airfoil mutant = new Airfoil();
                        AirfoilModifier.modAirfoil(mutant, a, b, r);

                        nextGen.removeIndividual(ind);
                        nextGen.addIndividual(mutant);
                  }
            }
      }

      public static void main(String[] args){
            GALite gal = new GALite(0.05,0.5,10,250);
            System.out.println("Best Overall: " +
gal.runOptimization());
      }
}
```

```
/**
 * Airfoil.java
 * Author: Brandon Morgan
 *
 * This class encapsulates all of the methods required to evaluate
the
 * fitness of a particular airfoil defined by parameters (A,B,R)
 *
 * A = xi coordinate of circle center in zeta plane
 * B = eta coordinate of circle center in zeta plane
 * R = radius of circle in zeta plane
 */

package galite;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;

import ann.ANNAirfoil;

import ga.Individual;


public class Airfoil implements Individual{
      public static final int FLUENT_EVALS = 200;
      public static int FUNCTION_EVALS = 0;
      //Note that these are kept public so updates to design
parameters
      //and fitness can be decoupled
      public double A,B,R;      // Design Parameters
      private double fitness;   // Store fitness so it only needs to
be calculated once

      /**Constructor for copying an existing airfoil**/
      public Airfoil(Airfoil af){
            this(af.A,af.B,af.R);
      }

      /**Default constructor.  Initialize everything to 0.**/
      public Airfoil(){
            A = 0;
            B = 0;
            R = 0;
            fitness = 0;
      }

      /**Main constructor**/
      public Airfoil(double A, double B, double R){
            this.A = A;
            this.B = B;
            this.R = R;
            setFitness();
      }

      /**Returns most positive intersection with xi axis**/
      public double getC(){
            return Math.sqrt(R*R - B*B) + A;
      }

      /**Returns most negative intersection with xi axis**/
      public  double getD(){
            return A - Math.sqrt(R*R - B*B);
      }

      /**Returns angle from xi axis to center of circle**/
      public double getDelta(){
            return Math.PI/2.0 + Math.atan(-A/B);
      }

      /**Returns distance from origin to center of circle**/
      public double getM(){
            return Math.sqrt(B*B + A*A);
      }

      /**Returns chord length (unscaled)**/
      public double getL(){
            double t = getThick();
            double c = getC();
            return 4.0 * (R - 0.77*t*c);
      }

      /**Returns camber**/
      public double getH(){
            double m = getM();
            double delta = getDelta();
            return 2.0*m*Math.sin(delta);
      }

      /**Returns thickness percentage (t/c)**/
      public double getThick(){
            double m = getM();
            double delta = getDelta();
            double c = getC();
            return -1.0/(0.77*c)*m*Math.cos(delta);
      }

      /**
```

```
       * Returns Cl based on potential theory for an angle of
attack, alpha
       * @param alpha angle of attack
       * @return coefficient of lift
       */
      public double getPotentialCl(double alpha){
            double t = getThick();
            double h = getH();
            double l = getL();
            return 2.0*Math.PI*(1.0 + 0.77*t)*Math.sin(alpha +
2.0*h/l);
      }

      /**This method should only be called when the airfoil is
constructed or
      when it is changed.  This is the method that will invoke
Fluent, ANN,
      potential theory, etc. to determine an airfoil's fitness.**/
      public void setFitness(){
            //Uncomment this line to use potential theory to get the
fitness
            //fitness = getPotentialCl(0);

            //Uncomment this line to use Fluent exclusively to get
the fitness
            //fitness = getScoreWithFluent();

            //Uncomment this line to use the ANN exclusively to get
the fitness
            //fitness = getScoreWithANN();

            //Uncomment this line to use Fluent/ANN to get the
fitness
            fitness = getScore();

      }

      /**This method simply returns the stored fitness value.**/
      public double getFitness(){
            return fitness;
      }

      /**
       * This method is responsible for carrying out the series of
events that
       * leads to invoking Fluent to evaluate the airfoil.
       *
       * First, a Gambit process is invoked which will read in
airfoil.dat and
       * create a mesh.  The java process blocks until this process
completes
       *
       * Second, a cleanup process is invoked to delete the
temporary files created
       * by Gambit.  The java process blocks until this process
completes.
```

```
	 *
	 * Third, the Fluent process is invoked.  Unfortunately, the
java process
	 * cannot block on this process because it spawns a separate
process of its
	 * own.  So, instead, the java process will spin until a
particular file,
	 * "trans.jou" is created by the Fluent process.  At this
time, the java
	 * process will continue and read in values from "cd-history"
and "cl-history"
	 * to determine the objective value.
	 *
	 * If the java process encounters a read error, the value of
the objective
	 * function returned will be -1.
	 *
	 * If the history files exist but are empty, the value of the
objective
	 * function returned will be -200
	 *
	 * @return objective value, as obtained with Fluent
	 */
	private double getScoreWithFluent(){
		try{
			Process gambitProc =
Runtime.getRuntime().exec("gambitTest.bat");
			gambitProc.waitFor();
			Process cleanupProc =
Runtime.getRuntime().exec("cleanup.bat");
			cleanupProc.waitFor();
			Process fluentProc =
Runtime.getRuntime().exec("fluentTest.bat");
			File transcript = new File("trans.jou");
			while(!transcript.exists()){
				//do nothing...we are waiting for fluent to
exit
			}

			BufferedReader cdInput =  new BufferedReader(new
FileReader(new File("cd-history")));
			BufferedReader clInput =  new BufferedReader(new
FileReader(new File("cl-history")));

			double cd = 1;
			double cl = -200;
			String line;

			//skip the first two lines
			for(int i=0;i<2;i++){
				cdInput.readLine();
				clInput.readLine();
			}

			//Uncomment this code to average over some number
of evaluations
```

```
                  /*int count = 0;
                  double sum = 0;
                  while(cdInput.ready()){
                        count++;
                        line = cdInput.readLine();
                        if(count > 1750){
                              line =
line.substring(line.indexOf("\t")+1);
                              sum += Double.parseDouble(line);
                        }
                        //line =
line.substring(line.indexOf("\t")+1);
                  }
                  if(sum != 0)
                        cd = sum/((double)(count - 1750.0));

                  count = 0;
                  sum = 0;
                  while(clInput.ready()){
                        count++;
                        line = clInput.readLine();
                        if(count > 1750){
                              line =
line.substring(line.indexOf("\t")+1);
                              sum += Double.parseDouble(line);
                        }
                  }
                  if(sum != 0)
                        cl = sum/((double)(count - 1750.0));*/

                  while(cdInput.ready()){
                        line = cdInput.readLine();
                        line = line.substring(line.indexOf("\t")+1);
                        cd = Double.parseDouble(line);
                  }

                  while(clInput.ready()){
                        line = clInput.readLine();
                        line = line.substring(line.indexOf("\t")+1);
                        cl = Double.parseDouble(line);
                  }

                  //return score cl or cl/cd with scaling for 0.1m
airfoil
                  double score = cl;//cl/cd;

                  return score;
            }
            catch(Exception e){
                  e.printStackTrace();
                  return -1;
            }

      }

      private double getScoreWithANN(){
            return GALite.net.evaluate(A,B,R);
      }

      public double getScore(){
            AirfoilPublisher.publishAirfoil(this,"airfoil.dat");
            try{

                  double score;

                        if(FUNCTION_EVALS < FLUENT_EVALS){
                                    score = getScoreWithFluent();
                                    ANNAirfoil foil = new
ANNAirfoil(A,B,R);
                                    foil.setObjective(score);
                                    GALite.net.addTrainingPt(foil);
                        }
                        else
                                    score = getScoreWithANN();

                  BufferedWriter recordWriter =  new
BufferedWriter(new FileWriter(new File("record.txt"), true));
                  if(FUNCTION_EVALS == FLUENT_EVALS){
                        System.out.println("Switching to ANN...");
                        recordWriter.write("Switching to ANN...\n");
                  }
                  recordWriter.write("(" + A + "," + B + "," + R +
") th: " + getThick() + " fitness: " + score +"\n");
                  recordWriter.close();

                  //System.out.println("fitness: " + cl/cd);
                  FUNCTION_EVALS++;
                  return score;
            }
            catch(Exception e){
                  e.printStackTrace();
                  return -1;
            }

      }

      public String toString(){
            return "(" + A + "," + B + "," + R + ") th: " +
getThick() + " fitness: " + getFitness();
      }

}
```

```
/**
 * AirfoilPublisher.java
 * Author: Brandon Morgan
 *
 * This class provides methods for getting a collection of points
describing
 * an Airfoil object.  (These methods used to be in Airfoil.java but
have
 * been moved here for clarity)
 */

package galite;


import java.awt.geom.Point2D;
import java.io.BufferedWriter;
import java.io.FileWriter;

public class AirfoilPublisher {
      public static final double eps = 1e-15; //Used for determining
absolute 0 and 1
      public static final int numPts = 50;    //Points to be
published on each surface

      /**
       * Convenience method for getting a set of points describing
the airfoil
       * @param af Airfoil
       * @param n number of points on each side
       * @return
       */
      public static Point2D.Double[] getAirfoilPoints(Airfoil af,
int n){
            return getAirfoilPoints(af.A,af.B,af.R,n);
      }

      /**
       * This method returns a collection of points that describe
the airfoil
       * defined by parameters (A,B,R)
       *
       * @param A xi-coordinate of circle center
       * @param B eta-coordinate of circle center
       * @param R radius of circle
       * @param n number of points on each surface of airfoil
       * @return array of 2*n points describing airfoil
       */
      public static Point2D.Double[] getAirfoilPoints(double
A,double B, double R,int n){
                  double c = Math.sqrt(R*R - B*B) + A;
                  double d = A - Math.sqrt(R*R - B*B);

                  double th_trailing = 1.5*Math.PI + Math.acos(B/R);
                  double th_leading = 1.5*Math.PI - Math.acos(B/R);

                  double translation = d * (1.0 + Math.pow(c,2.0) /(
Math.pow(d,2)+ Math.pow(0,2) ) );
```

```
                  double scaleFactor = 1.0 / (c * (1.0 +
Math.pow(c,2.0) /( Math.pow(c,2)+ Math.pow(0,2) ) ) - d * (1.0 +
Math.pow(c,2.0) /( Math.pow(d,2)+ Math.pow(0,2) ) ));

                  double[] airfoil_x = new double[2*n];
                  double[] airfoil_y = new double[2*n];

                  //leading edge to trailing edge --> d to c, top
surface
                  double theta,xi,eta;
                  //System.out.println("NEW AIRFOIL!!! leading: " +
Airfoil.rad2Deg(th_leading) + "trailing: " +
Airfoil.rad2Deg(th_trailing));
                  for(int i=0;i<n;i++){
                        theta = th_leading - i * (th_leading +
2.0*Math.PI-th_trailing)/(n-1);
                        xi = Math.cos(theta)*R+A;
                  eta = Math.sin(theta)*R+B;
                  airfoil_x[i] = ( xi * (1.0 + Math.pow(c,2.0) /(
Math.pow(xi,2)+ Math.pow(eta,2) ) ) - translation)*scaleFactor;
                  if(Math.abs(airfoil_x[i]) < eps || Math.abs(1.0-
airfoil_x[i]) < eps)
                        airfoil_x[i] = Math.round(airfoil_x[i]);
                  airfoil_y[i] = ( eta * (1.0 - Math.pow(c, 2.0) /(
Math.pow(xi,2)+ Math.pow(eta,2) ) ) )*scaleFactor;
                  if(Math.abs(airfoil_y[i]) < eps || Math.abs(1.0-
airfoil_y[i]) < eps)
                        airfoil_y[i] = Math.round(airfoil_y[i]);
                  }

                  //leading edge to trailing edge --> d to c, bottom
surface
                  for(int i=0;i<n;i++){
                        theta = th_leading + i * (th_trailing-
th_leading)/(n-1);
                        xi = Math.cos(theta)*R+A;
                  eta = Math.sin(theta)*R+B;
                  airfoil_x[i+n] = ( xi * (1.0 + Math.pow(c,2.0) /(
Math.pow(xi,2)+ Math.pow(eta,2) ) ) - translation)*scaleFactor;
                  if(Math.abs(airfoil_x[i+n]) < eps || Math.abs(1.0-
airfoil_x[i+n]) < eps)
                        airfoil_x[i+n] = Math.round(airfoil_x[i+n]);
                  airfoil_y[i+n] = ( eta * (1.0 - Math.pow(c, 2.0)
/( Math.pow(xi,2)+ Math.pow(eta,2) ) ) )*scaleFactor;
                  if(Math.abs(airfoil_y[i+n]) < eps || Math.abs(1.0-
airfoil_y[i+n]) < eps)
                        airfoil_y[i+n] = Math.round(airfoil_y[i+n]);
                  }

                  double maxVal = 0;
                  for(int i=0;i<2*n;i++){
                        if(Math.abs(airfoil_x[i]) > maxVal)
                              maxVal = airfoil_x[i];
                        if(Math.abs(airfoil_y[i]) > maxVal)
                              maxVal = airfoil_y[i];
                  }
                  for(int i=0;i<2*n;i++){
                        airfoil_x[i] /= maxVal;
                        airfoil_y[i] /= maxVal;
                  }

                  Point2D.Double[] ret = new Point2D.Double[2*n];
                  for(int i=0;i<2*n;i++){
                        ret[i] = new
Point2D.Double(airfoil_x[i],airfoil_y[i]);
                  }

                  return ret;

      }

      /**
       * Convenience method for publishing a file containing an
airfoil description
       * @param af Airfoil
       * @param filename name of file
       * @return true if file was written, false otherwise
       */
      public static boolean publishAirfoil(Airfoil af, String
filename){
            return publishAirfoil(af.A,af.B,af.R,filename);
      }

      /**
       * This method writes a file containing a collection of points
that describe
       * an airfoil parameterized by (A,B,R).  [For use with Gambit]
       * @param A xi-coordinate of circle center
       * @param B eta-coordinate of circle center
       * @param R radius of circle
       * @param filename name of file
       * @return true if file was written, false otherwise
       */
      public static boolean publishAirfoil(double A, double B,
double R, String filename){
            Point2D.Double[] pts = getAirfoilPoints(A,B,R,numPts);
            try{
                // Create file
                FileWriter fstream = new FileWriter(filename);
                BufferedWriter out = new BufferedWriter(fstream);
                out.write(numPts+ " 2\n");
                for(int i=0;i<2*numPts;i++){
                  out.write(pts[i].x+ " " + pts[i].y + " 0\n");
                }
                out.close();
                return true;
        }
        catch (Exception e){
              System.err.println("Error: " + e.getMessage());
              return false;
        }
      }
```

```
}
```

```
/**
 * AirfoilModifier.java
 * Author: Brandon Morgan
 *
 * This class provides encapsulation of all the checks that must
occur
 * when an airfoil is to be created or changed to ensure that
 * the constraints are met.
 */

package galite;

public class AirfoilModifier {
      public static final double minA = -10;
      public static final double maxA = -.35;
      public static final double minB = 0;
      public static final double maxB = 10;
      public static final double maxThck = 0.20;
      public static final double minThck = 0.01;

      public static void modAirfoil(Airfoil af, double aNew, double
bNew, double rNew){
            //Check A: minA <= A <= maxA
            if(aNew < minA)
                  af.A = minA;
            else if(aNew > maxA)
                  af.A = maxA;
            else
                  af.A = aNew;

            //Check B: minB <= B <= maxB
            if(bNew < minB)
                  af.B = minB;
            else if(bNew > maxB)
                  af.B = maxB;
            else
                  af.B = bNew;

            //Check R: C must be strictly positive
            af.R = rNew;
            if(af.getC() <= 0 || Double.isNaN(af.getC()))
                  af.R = Math.sqrt(Math.pow(1.01*af.A,2.0) +
Math.pow(af.B, 2.0));

            //Now check thickness (this is also checking R)
            if(af.getThick() > maxThck)
```

```
                  af.R = Math.sqrt( Math.pow(-1.0/(0.77*maxThck) *
af.getM()*Math.cos(af.getDelta()) - af.A,2.0) + Math.pow(af.B,2.0)
);
            else if(af.getThick() < minThck)
                  af.R = Math.sqrt( Math.pow(-1.0/(0.77*minThck) *
af.getM()*Math.cos(af.getDelta()) - af.A,2.0) + Math.pow(af.B,2.0)
);

            af.setFitness();
      }
}
```

```
package ga;

/*
 * @(#)BubbleSortAlgorithm.java     1.6 95/01/31 James Gosling
 *
 * Copyright (c) 1994 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON-COMMERCIAL purposes and without
 * fee is hereby granted provided that this copyright notice
 * appears in all copies. Please refer to the file "copyright.html"
 * for further important copyright and licensing information.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
OF
 * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
LIMITED
 * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE
FOR
 * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 */

/**
 * A bubble sort demonstration algorithm
 * SortAlgorithm.java, Thu Oct 27 10:32:35 1994
 *
 * @author James Gosling
 * @version       1.6, 31 Jan 1995
 *
 * Modified 23 Jun 1995 by Jason Harrison@cs.ubc.ca:
 *   Algorithm completes early when no items have been swapped in
the
 *   last pass.
 */
public class BubbleSorter{
    public static void sort(Individual a[]){
      for (int i = a.length; --i>=0; ) {
            boolean flipped = false;
            for (int j = 0; j<i; j++) {

                  if (a[j].getFitness() > a[j+1].getFitness()) {
                        Individual T = a[j];
                        a[j] = a[j+1];
                        a[j+1] = T;
                        flipped = true;
                  }
            }
            if (!flipped) {
                  return;
            }
        }
    }
}
```

```
/**
 * Generation.java
 * Author: Brandon Morgan
 *
 * This class provides a data structure for storing all of the
Individual members
 * of a GA Generation
 */

package ga;

import java.util.Iterator;
import java.util.Vector;


public class Generation {
      private Vector<Individual> inds;
      private int totalNum;

      public Generation(int totalNum){
            this.totalNum = totalNum;
            inds = new Vector<Individual>();
      }

      public boolean addIndividual(Individual i){
            if(inds.size() < totalNum)
                  return inds.add(i);
            else
                  return false;
	}

	public double getGenerationFitness(){
		double ret = 0;
		Iterator<Individual> it = inds.iterator();
		double fit;
		while(it.hasNext()){
			fit = it.next().getFitness();
			if(fit > 0)
				ret += fit;
		}
		return ret;
	}

	public boolean removeIndividual(Individual i){
		return inds.remove(i);
	}

	public Individual removeIndividual(int i){
		return inds.remove(i);
	}

	public boolean isFull(){
		return inds.size() >= totalNum;
	}

	public Individual getBestIndividual(){
		Iterator<Individual> it = inds.iterator();
		Individual best = null;

		while(it.hasNext()){
			Individual ind = it.next();
			if(best == null || ind.getFitness() >
best.getFitness())
				best = ind;
		}

		return best;
	}

	public int getTotalNum(){
		return totalNum;
	}

	public Individual getIndividual(int i){
		return inds.get(i);
	}

	public int getCount(){
		return inds.size();
	}

	public Individual[] getIndividuals(){
		return inds.toArray(new Individual[0]);
	}
	public void removeIndividuals(int num){
		Individual[] indArray = inds.toArray(new Individual[0]);
		BubbleSorter.sort(indArray);
		for(int i=0;i<num;i++){
			removeIndividual(indArray[i]);
		}
	}

	public String toString(){
		Individual[] indArray = inds.toArray(new Individual[0]);
		BubbleSorter.sort(indArray);
		String str = "";

		for(int i=0;i<indArray.length;i++){
			Individual ind = indArray[i];
			str += "" + i + ": " + ind.getFitness() + "\n";
		}

		return str;
	}
}
```

```
/**
 * Individual.java
 * Author: Brandon Morgan
 *
 * Interface for use with GA
 */

package ga;

public interface Individual {
	public double getFitness();
}
```

```
/**
 * ANNAirfoil.java
 * Author: Brandon Morgan
 *
 * This class provides encapsulation of airfoil data for the ANN
package
 * Note that data is stored in a scaled form - that is: a, b, & r
are all
 * scaled to be between -1 and 1.
 */

package ann;

import galite.AirfoilModifier;

public class ANNAirfoil {
      private double a,b,r;
      private double objective;

      public ANNAirfoil(double a, double b, double r){
            double maxA = Math.abs(AirfoilModifier.minA);  //maximum
magnitude of A
            double maxB = Math.abs(AirfoilModifier.maxB);  //maximum
magnitude of B
            double maxM = Math.sqrt(maxB*maxB + maxA*maxA);
//maximum magnitude of M
            double maxDelta = Math.PI; //maximum magnitude of delta
            double maxR = Math.sqrt( Math.pow(-
1.0/(0.77*AirfoilModifier.minThck) * maxM*Math.cos(maxDelta) +
maxA,2.0) + Math.pow(maxB,2.0) );

            // These values must be between -1 and 1, hence the
scaling
            this.a = a/maxA;
            this.b = b/maxB;
            this.r = r/maxR;
      }

      public void setObjective(double o){
            objective = o;
      }

      public double getObjective(){
            return objective;
      }

      public double getA(){
            return a;
      }

      public double getB(){
            return b;
      }

      public double getR(){
            return r;
      }

      public ANNVector getInputVector(){
            double[] ret = {a,b,r};
            return new ANNVector(ret);
      }

      public ANNVector getOutputVector(){
            double[] ret = {objective};
```

```
		return new ANNVector(ret);
	}

	public String toString(){
		return "(" + a + ", " + b + ", " + r + "): " +
objective;
	}
}
```

```
/**
 * ANNMatrix.java
 * Author: Brandon Morgan
 *
 * This class general matrix operations for use with training the
ANN
 *
 */


package ann;

public class ANNMatrix {
	public ANNVector[] data;

	public ANNMatrix(ANNVector[] data){
		this.data = data.clone();
	}

	public ANNMatrix times(double c){
		ANNVector[] ret = new ANNVector[data.length];
		for(int i=0;i<ret.length;i++){
			ret[i] = data[i].times(c);
		}
		return new ANNMatrix(ret);
	}

	public ANNVector times(ANNVector vec) throws
ANNVector.VectorLengthException{
		double[] ret = new double[vec.length()];
		for(int i=0;i<ret.length;i++){
			ret[i] = data[i].dot(vec);
		}
		return new ANNVector(ret);
	}

	 public ANNMatrix plus(ANNMatrix mat) throws
ANNVector.VectorLengthException{
		ANNVector[] ret = new ANNVector[data.length];
		for(int i=0;i<ret.length;i++){
```

```
                ret[i] = data[i].plus(mat.getRow(i));
          }
          return new ANNMatrix(ret);
    }

     public ANNMatrix minus(ANNMatrix mat) throws
ANNVector.VectorLengthException{
                ANNVector[] ret = new ANNVector[data.length];
                for(int i=0;i<ret.length;i++){
                      ret[i] = data[i].minus(mat.getRow(i));
                }
                return new ANNMatrix(ret);
          }

    public double max(){
          double ret = data[0].max();
          for(int i=1;i<data.length;i++){
                double mx = data[i].max();
                if(mx > ret)
                      ret = mx;
          }
          return ret;
    }

    public double min(){
          double ret = data[0].min();
          for(int i=1;i<data.length;i++){
                double mx = data[i].min();
                if(mx < ret)
                      ret = mx;
          }
          return ret;
    }

     public ANNVector getRow(int i){
           return new ANNVector(data[i].data);
     }

     public ANNVector getCol(int c){
           double[] ret = new double[data.length];
           for(int i=0;i<data.length;i++){
                 ret[i] = data[i].get(c);
           }
           return new ANNVector(ret);
     }

     public String toString(){
           String str = "";
           for(int i=0;i<data.length;i++){
                 str += data[i];
                 str += "\n";
           }
           return str;
     }

     public static ANNMatrix abs(ANNMatrix mat){
            ANNVector[] ret = new ANNVector[mat.data.length];
            for(int i=0;i<ret.length;i++){
                  ret[i] = ANNVector.abs(mat.data[i]);
            }
            return new ANNMatrix(ret);
      }
}
```

```
/**
 * ANNTrainer.java
 * Author: Brandon Morgan
 *
 * This is the base class used to train an ANN.  I have stripped
away much of
 * the implementation that was used for training offline.  This file
remains
 * because OnlineANN inherits from it
 */

package ann;/*

 */

import java.io.IOException;
import java.util.Random;
import java.util.Vector;


public class ANNTrainer {
      public static final double rand_min = -0.01;
      public static final double rand_max = 0.01;
      //public static final double eta = 0.0005; //Cl/Cd
      //public static final double eps = 0.005; //Cl/Cd
      public static final double eta = .5; //Cl
      public static final double eps = 0.005; //Cl
      public static final Random numGen = new Random(2182);

      protected int H; //Number of hidden units
      protected int D; //Number of input units
      protected int K; //Number of output units

      ANNMatrix w,w_last,del_w; //input weights, last time step,
update step
      ANNMatrix v,v_last,del_v; //hidden weights, last time step,
update step
      ANNVector z,y,x,r; //hidden unit evaluations, output, input,
true output
      Vector<ANNAirfoil> data; //training data
      public ANNTrainer(int H) throws IOException,
ANNVector.VectorLengthException{
            this.H = H;
            D = 3;
            K = 1;

            //initialize();
            //train();
      }

      protected void train() throws ANNVector.VectorLengthException{

            double error = 1;//abs(w.minus(w_last)).max() +
abs(v.minus(v_last)).max();

            while( error > eps){
                  //System.out.println("Error ("+ H + "): " +
error);
                  v_last = new ANNMatrix(v.data);
                  w_last = new ANNMatrix(w.data);

                  for(int t=0;t<data.size();t++){
                        x = data.get(t).getInputVector();
                        r = data.get(t).getOutputVector();

                        for(int h=0;h<H;h++){
                              z.data[h] = sigmoid(w.data[h].dot(x));
                        }

                        for(int i=0;i<K;i++){
                              y.data[i] = v.data[i].dot(z);
                        }

                        for(int i=0;i<K;i++){
                              del_v.data[i] = z.times((r.data[i]-
y.data[i])).times(eta);
                        }

                        for(int h=0;h<H;h++){
                              double sumDiff = 0;
                              for(int i=0;i<K;i++){
                                    sumDiff += (r.data[i]-
y.data[i])*v.data[i].data[h];
                              }
                              del_w.data[h] =
x.times(eta*sumDiff*z.data[h]*(1-z.data[h]));
                        }

                        for(int i=0;i<K;i++){
                              v.data[i] =
v.data[i].plus(del_v.data[i]);
                        }
                        for(int h=0;h<H;h++){
                              w.data[h] =
w.data[h].plus(del_w.data[h]);
                        }
                  }

                  error = abs(w.minus(w_last)).max() +
abs(v.minus(v_last)).max();

            }

      }

      //Convenience method
      static ANNVector abs(ANNVector v){
            return ANNVector.abs(v);
      }

      //Convenience method
      static ANNMatrix abs(ANNMatrix m){
            return ANNMatrix.abs(m);
      }

      static double[] ones(int k){
            double[] ret = new double[k];
            for (int i=0;i<ret.length;i++){
                  ret[i] = 1;
            }
            return ret;
      }

      static double[] zeros(int k){
            double[] ret = new double[k];
            for (int i=0;i<ret.length;i++){
                  ret[i] = 0;
            }
            return ret;
      }

      static double[] rands(int k, double min, double max){
            double[] ret = new double[k];
            for(int i=0;i<ret.length;i++){
                  //ret[i] = min + Math.random()*(max-min);
                  ret[i] = min + numGen.nextDouble()*(max-min);
            }
            return ret;
      }

      static double sigmoid(double t){
            return 1.0 / (1.0+Math.exp(-t));
      }


}
```

```
/**
 * ANNVector.java
 * Author: Brandon Morgan
 *
 * This class general vector operations for use with training the
ANN
 *
 */

package ann;

public class ANNVector {
      public double[] data;

      public ANNVector(double[] data){
            this.data = data.clone();
      }

      public double dot(ANNVector v) throws VectorLengthException{
            return dot(v.data);
      }

      public double dot(double[] v) throws VectorLengthException{
            if(v.length != data.length)
                  throw new VectorLengthException("Vectors must be
same length");
            else{
                  double sum = 0;
                  double[] d = new double[data.length];
                  for(int i=0;i<d.length;i++){
                        sum += data[i]*v[i];
                  }
                  return sum;
            }
      }

      public ANNVector plus(ANNVector v) throws
VectorLengthException{
            return plus(v.data);
      }

      public ANNVector plus(double[] v) throws
VectorLengthException{
            if(v.length != data.length)
                  throw new VectorLengthException("Vectors must be
same length");
            else{
                  double[] d = new double[v.length];
                  for(int i=0;i<d.length;i++){
                        d[i] = data[i]+v[i];
                  }
                  return new ANNVector(d);
            }
      }
      public ANNVector minus(ANNVector v) throws
VectorLengthException{
            return minus(v.data);
      }

      public ANNVector minus(double[] v) throws
VectorLengthException{
            if(v.length != data.length)
                  throw new VectorLengthException("Vectors must be
same length");
            else{
                  double[] d = new double[v.length];
                  for(int i=0;i<d.length;i++){
                        d[i] = data[i]-v[i];
                  }
                  return new ANNVector(d);
            }
      }

      public ANNVector times(ANNVector v) throws
VectorLengthException{
            return times(v.data);
      }

      public ANNVector times(double[] v) throws
VectorLengthException{
            if(v.length != data.length)
                  throw new VectorLengthException("Vectors must be
same length");
            else{
                  double[] d = new double[v.length];
                  for(int i=0;i<d.length;i++){
                        d[i] = data[i]*v[i];
                  }
                  return new ANNVector(d);
            }
      }

      public ANNVector pow(double c){
            double[] d = new double[data.length];
            for(int i=0;i<d.length;i++){
                  d[i] = Math.pow(data[i],c);
            }
            return new ANNVector(d);
      }

      public ANNVector times(double c){
            double[] d = new double[data.length];
            for(int i=0;i<d.length;i++){
                  d[i] = data[i]*c;
            }
            return new ANNVector(d);
      }

      public String toString(){
            String str = "<";
            for(int i=0;i<data.length;i++){
                  str += data[i];
                  if(i!= data.length -1)
                        str += ", ";
            }
            str += ">";
            return str;
      }

      public double max(){
            double ret = data[0];
            for(int i=1;i<data.length;i++){
                  if(data[i] > ret)
                        ret = data[i];
            }
            return ret;
      }

      public int maxIndex(){
            int ret = 0;
            double maxVal = data[0];
            for(int i=1;i<data.length;i++){
                  if(data[i] > maxVal){
                        ret = i;
                        maxVal = data[i];
                  }
            }
            return ret;
      }

      public double min(){
            double ret = data[0];
            for(int i=1;i<data.length;i++){
                  if(data[i] < ret)
                        ret = data[i];
            }
            return ret;
      }

      public int minIndex(){
            int ret = 0;
            double minVal = data[0];
            for(int i=1;i<data.length;i++){
                  if(data[i] < minVal){
                        minVal = data[i];
                        ret = i;
                  }
            }
            return ret;
      }

      public double get(int i){
            return data[i];
      }
```

```
      public int length(){
            return data.length;
      }

      public static ANNVector abs(ANNVector vec){
            double[] dat = new double[vec.data.length];
            for(int i=0;i<dat.length;i++){
                  dat[i] = Math.abs(vec.data[i]);
            }
            return new ANNVector(dat);
      }

      public class VectorLengthException extends Exception {
            public VectorLengthException(String s){
                  super(s);
            }
      }

}
```

```
/**
 * OnlineANN.java
 * Author: Brandon Morgan
 *
 * This class trains an ANN online.  It is used by GALite when
running in
 * GA/ANN mode.
 */

package ann;

import galite.Airfoil;
import galite.GALite;

import java.io.IOException;
import java.util.Vector;


public class OnlineANN extends ANNTrainer{

      public OnlineANN(int H) throws IOException,
ANNVector.VectorLengthException{
            super(H);
            initialize();
      }

      protected void initialize() throws IOException{
            ANNVector[] w_vec = new ANNVector[H];
            ANNVector[] w_last_vec = new ANNVector[H];
            ANNVector[] del_w_vec = new ANNVector[H];
            for(int i=0;i<H;i++){
                  w_vec[i] = new
ANNVector(rands(D,rand_min,rand_max));
                  w_last_vec[i] = new ANNVector(ones(D));
                  del_w_vec[i] = new ANNVector(zeros(D));
            }
            w = new ANNMatrix(w_vec);
            w_last = new ANNMatrix(w_last_vec);
            del_w = new ANNMatrix(del_w_vec);

            ANNVector[] v_vec = new ANNVector[K];
            ANNVector[] v_last_vec = new ANNVector[K];
            ANNVector[] del_v_vec = new ANNVector[K];
            for(int i=0;i<K;i++){
                  v_vec[i] = new
ANNVector(rands(H,rand_min,rand_max));
                  v_last_vec[i] = new ANNVector(ones(H));
                  del_v_vec[i] = new ANNVector(zeros(H));
            }
            v = new ANNMatrix(v_vec);
            v_last = new ANNMatrix(v_last_vec);
            del_v = new ANNMatrix(del_v_vec);

            z = new ANNVector(zeros(H));
            y = new ANNVector(zeros(K));
            x = new ANNVector(zeros(D));

            data = new Vector<ANNAirfoil>();
      }

      //convenience method for interfacing with GAirfoils
      public double evaluate(double A, double B, double R){
            ANNAirfoil af = new ANNAirfoil(A,B,R);
            double[] x = new double[]
{af.getA(),af.getB(),af.getR()};
            try{
                  ANNVector y = evaluate(new ANNVector(x));
                  return y.data[0];
            }
            catch(ANNVector.VectorLengthException vle){
                  vle.printStackTrace();
                  return -300;
            }

      }

      public ANNVector evaluate(ANNVector x) throws
ANNVector.VectorLengthException{
            ANNVector z = new ANNVector(ANNTrainer.zeros(H));
            ANNVector y = new ANNVector(ANNTrainer.zeros(K));

            for(int h=0;h<H;h++){
                  z.data[h] = ANNTrainer.sigmoid(w.data[h].dot(x));
            }

            for(int i=0;i<K;i++){
                  y.data[i] = v.data[i].dot(z);
            }

            return y;
      }

      public void addTrainingPt(ANNAirfoil foil){
            data.add(foil);
            try{
                  ANNVector[] w_vec = new ANNVector[H];
                  ANNVector[] w_last_vec = new ANNVector[H];
                  ANNVector[] del_w_vec = new ANNVector[H];
                  for(int i=0;i<H;i++){
                        w_vec[i] = new
ANNVector(rands(D,rand_min,rand_max));
                        w_last_vec[i] = new ANNVector(ones(D));
                        del_w_vec[i] = new ANNVector(zeros(D));
                  }
                  w = new ANNMatrix(w_vec);
                  w_last = new ANNMatrix(w_last_vec);
                  del_w = new ANNMatrix(del_w_vec);

                  ANNVector[] v_vec = new ANNVector[K];
                  ANNVector[] v_last_vec = new ANNVector[K];
                  ANNVector[] del_v_vec = new ANNVector[K];
                  for(int i=0;i<K;i++){
                        v_vec[i] = new
ANNVector(rands(H,rand_min,rand_max));
                        v_last_vec[i] = new ANNVector(ones(H));
                        del_v_vec[i] = new ANNVector(zeros(H));
                  }
                  v = new ANNMatrix(v_vec);
                  v_last = new ANNMatrix(v_last_vec);
                  del_v = new ANNMatrix(del_v_vec);

                  z = new ANNVector(zeros(H));
                  y = new ANNVector(zeros(K));
                  x = new ANNVector(zeros(D));
                  if(GALite.DEBUG_LEVEL>1)
                        System.out.print("Training point (" +
Airfoil.FUNCTION_EVALS + ")...");
                  train();
                  if(GALite.DEBUG_LEVEL>1)
                        System.out.print("Complete!\n");
            }
            catch(ANNVector.VectorLengthException vle){
                  vle.printStackTrace();
                  System.exit(-1);
            }
      }



}
```

Appendix B

Java Code for Truck Front Shape Optimization Using a Genetic Algorithm

```
/**
 * GATruck_Miles.java
 * Author: Miles Bellman
 *
 * This is the main class used in optimization.
 */

package gatruck_Miles;


public class GATruck_Miles {
      private int genSize, numGens, E;
      private double removePercentage, mutRate;
      public static TruckMaker bestTruck;
      private Generation one;
      public static double timeStep = 0.02;
      private int existingGenerations = 0;


      public GATruck_Miles(int genSize, int numGens, double
removePercentage, double mutRate){
            this.genSize = genSize;
            this.numGens = numGens;
            this.removePercentage = removePercentage;
            E = (int) Math.round(genSize * this.removePercentage);
            this.mutRate = mutRate;
      }

      private TruckMaker generateIndividual(){
            /**random bezier coordinates**/
            /*double x0 = Math.random() * (Truck.minX-Truck.maxX) +
Truck.maxX;
            //System.out.println("x0: " + x0);*/
            double x1 = Math.random() * (Truck.minX-Truck.maxX) +
Truck.maxX;

                  //Check to make sure x1 > x0. If not, let x2 be
randomly
                  //between x0 and maxX.
                  /*if(x1 > x0){
                        x1 = x1;
                  }
                  else{
                        x1 = Math.random() * (x0-Truck.maxX) +
Truck.maxX;
                  }*/

            //System.out.println("x1: " + x1);
            double y1 = Math.random() * (Truck.maxY-Truck.minY) +
Truck.minY;
            //System.out.println("y1: " + y1);
            double x2 = Math.random() * (Truck.minX-Truck.maxX) +
Truck.maxX;

                  //Check to make sure x2 > x1. If not, let x2 be
randomly
                  //between x1 and maxX.
                  /*if(x2 > x1){
                        x2 = x2;
                  }
                  else{
                        x2 = Math.random() * (x1-Truck.maxX) +
Truck.maxX;
                  }*/

            //System.out.println("x2: " + x2);
            double y2 = Math.random() * (Truck.maxY-Truck.minY) +
Truck.minY;
            //System.out.println("y2: " + y2);
            /*double x3 = Math.random() * (Truck.minX-Truck.maxX) +
Truck.maxX;

                  //Check to make sure x3 > x2. If not, let x2 be
randomly
                  //between x0 and maxX.
                  if(x3 > x2){
                        x3 = x3;
                  }
                  else{
                        x3 = Math.random() * (x2-Truck.maxX) +
Truck.maxX;
                  }

            //System.out.println("x3: " + x3);*/

            //create new truck
            /*TruckMaker truck = new TruckMaker(x0, x1, y1, x2, y2,
x3);*/

            TruckMaker truck = new TruckMaker(x1, y1, x2, y2);

            //check for truck front boundaries
            Truck.modTruck(truck);

            return truck;
      }
      /**Run the optimization*/
      public void runOptimization(){
            //one = manyIndividuals(); //Use this if starting from
the beginning

            one = existingIndividuals(7); //Input existing truck
generation if starting from the middle. Be sure to manually put in
truck parameters in existingIndividuals() method

            bestTruck = new TruckMaker(); //Create an initial bad
fitness truck to compare the first trucks against

            for (int i=1+existingGenerations; i<numGens; i++){
                  one.determineFitness(i);
//
//                /**Find truck with lowest coefficient of drag */
                  System.out.println("***** Generation " + i +
"*****\n");
                  bestTruck = one.getBestTruck();
                  System.out.println("Best truck cd = " +
bestTruck.getFitness() + "\n");
//
//                /**Create new generations to find best truck*/
                  advanceGen();
            }

      }

      /**generate genSize number of individuals**/
      private Generation manyIndividuals(){
            //new generation of trucks
            Generation trucks = new Generation(genSize);

            //loop to generate 10 trucks
            for(int i=0;i<genSize;i++){
                  trucks.addTruck(generateIndividual());
            }

            return trucks;
      }

      /**create generation with existing trucks**/
      private Generation existingIndividuals(int
existingGenerations){
            //Input the generation
            this.existingGenerations = existingGenerations-1;
//subtract one because the for loop in runOptimization adds 1

            //new generation of trucks
            Generation trucks = new Generation(genSize);

            //add existing trucks manually
            trucks.addTruck(new TruckMaker(-2.9948310257474904,-
0.222,-0.95,-0.1267490592651832,1.4204726));
```

```
            trucks.addTruck(new TruckMaker(-3.0,-0.222,-0.95,-
0.08525626657512106,1.3125177));
            trucks.addTruck(new TruckMaker(-2.514424824276382,-
0.222,-1.686611227710031,-0.222,1.0905675));
            trucks.addTruck(new TruckMaker(-3.0,-0.222,-
1.7084102276527584,-0.222,1.2414024));
            trucks.addTruck(new TruckMaker(-3.0,-0.222,-
1.444037409809154,-0.17239877085533709,1.1084004));
            trucks.addTruck(new TruckMaker(-3.0,-0.222,-
1.5220690038020037,-0.222,1.1736187));
            trucks.addTruck(new TruckMaker(-0.95,-
0.053305132261264786,-
1.8052743015662764,0.06275919120570522,1.1253602));
            trucks.addTruck(new TruckMaker(-1.0977770676437586,-
0.01600149800927278,-
1.9410896745427835,0.06680081185688938,1.0992951));
            trucks.addTruck(new TruckMaker(-2.217934867076381,-
0.18339685691539798,-1.6518612234587944,-
0.0716736455931544,1.0832282));
            trucks.addTruck(new TruckMaker(-3.0,-0.222,-
1.6340503593536528,-0.222,1.0769502));

            return trucks;
      }
      /**Use previous generations to create new generations*/
      private void advanceGen(){
            //System.out.println("Begin advance gen\n");
            Generation nextGen = new Generation(genSize);

            /**Create E number of new trucks by using coordinates
from two random trucks from the previous generation*/
            int count = 0;
            while(count<E){
                  TruckMaker truck1 =
one.getTruck((int)Math.floor(Math.random()*genSize));
                  TruckMaker truck2 =
one.getTruck((int)Math.floor(Math.random()*genSize));
                  if(truck1 != truck2){
                        nextGen.addTruck(crossover(truck1,truck2));
                        count++;
                  }
            }

                  /**Check to make sure it works*/
                  //System.out.print("***Next generation of
trucks***\n");
                  //nextGen.outputTrucks();

            /**Remove E number of trucks from the previous
generation with the highest Cd*/
            naturalSelection();
            //one.outputTrucks();
            //System.out.println(one.getTruckVectorSize());

            /**Add surviving trucks to nextGen of trucks*/
            for(int i=0;i<one.getTruckVectorSize();i++){
                  nextGen.addTruck(one.getTruck(i));
            }

                  /**Check to make sure it works*/
                  //System.out.println("***Next generation of trucks
with survivors***\n");
                  //nextGen.outputTrucks();

            /**Mutate*/
            mutate(nextGen);

            /**Original generation becomes nextGen*/
            one = nextGen;
            //System.out.println("***Next generation of trucks***");
            //one.outputTrucks();


      }

      /**Take coordinates of two trucks and combines them to get a
new truck*/
      private TruckMaker crossover(TruckMaker truck1, TruckMaker
truck2){
            //System.out.println("Begin crossover\n");
            //double x0_1,x0_2;
            double x1_1,x1_2;
            double y1_1,y1_2;
            double x2_1,x2_2;
            double y2_1,y2_2;
            //double x3_1,x3_2;

            if(truck1.getFitness() < truck2.getFitness()){
                  //x0_2 = truck1.X0;
                  x1_2 = truck1.X1;
                  y1_2 = truck1.Y1;
                  x2_2 = truck1.X2;
                  y2_2 = truck1.Y2;
                  //x3_2 = truck1.X3;
                  //x0_1 = truck2.X0;
                  x1_1 = truck2.X1;
                  y1_1 = truck2.Y1;
                  x2_1 = truck2.X2;
                  y2_1 = truck2.Y2;
                  //x3_1 = truck2.X3;
            } else {
                  //x0_1 = truck1.X0;
                  x1_1 = truck1.X1;
                  y1_1 = truck1.Y1;
                  x2_1 = truck1.X2;
                  y2_1 = truck1.Y2;
                  //x3_1 = truck1.X3;
                  //x0_2 = truck2.X0;
                  x1_2 = truck2.X1;
                  y1_2 = truck2.Y1;
                  x2_2 = truck2.X2;
                  y2_2 = truck2.Y2;
                  //x3_2 = truck2.X3;
            }

            /**Create new truck coordinates with crossover of two
old trucks biasing towards truck with smaller Cd*/
            //double x0 = Math.random()*(x0_2-x0_1) + x0_2;
            double x1 = Math.random()*(x1_2-x1_1) + x1_2;
            double y1 = Math.random()*(y1_2-y1_1) + y1_2;
            double x2 = Math.random()*(x2_2-x2_1) + x2_2;
            double y2 = Math.random()*(y2_2-y2_1) + y2_2;
            //double x3 = Math.random()*(x3_2-x3_1) + x3_2;

            TruckMaker truck = new TruckMaker(x1,y1,x2,y2);
            Truck.modTruck(truck);

            return truck;
      }

      private void naturalSelection(){
            //System.out.println("***Begin natural selection***\n");
            BubbleSorter.sort(one);
            //System.out.println("***Sorted Trucks from worst to
best***\n");
            //one.outputTrucks();
            one.removeTrucks(genSize-E);
            //System.out.println("***Remaining trucks after natural
selection\n***");
            //one.outputTrucks();
      }

      private void mutate(Generation nextGen){
            for(int i=0;i<nextGen.getGenSize();i++){
                  TruckMaker truck = nextGen.getTruck(i);
                  truck.fitness = 1000000; //make sure truck with
new coordinates is evaluated and not assumed to have its old fitness
value
                  if(Math.random() <= mutRate) {

                        //double x0 = Math.random() * (Truck.minX-
Truck.maxX) + Truck.maxX;
                        //System.out.println("x0: " + x0);
                        double x1 = Math.random() * (Truck.minX-
Truck.maxX) + Truck.maxX;
                        //System.out.println("x1: " + x1);
                        double y1 = Math.random() * (Truck.maxY-
Truck.minY) + Truck.minY;
                        //System.out.println("y1: " + y1);
                        double x2 = Math.random() * (Truck.minX-
Truck.maxX) + Truck.maxX;
                        //System.out.println("x2: " + x2);
                        double y2 = Math.random() * (Truck.maxY-
Truck.minY) + Truck.minY;
                        //System.out.println("y2: " + y2);
                        //double x3 = Math.random() * (Truck.minX-
Truck.maxX) + Truck.maxX;
```

```
                        TruckMaker mutant = new
TruckMaker(x1,y1,x2,y2);
                        Truck.modTruck(mutant);

                        nextGen.removeTruck(truck);
                        //System.out.println("Truck removed for
mutation");
                        nextGen.addTruck(mutant);
                        //System.out.println("Truck added for
mutation");
                  }
            }
      }



//    private convertToGambitTrucks() {
//          //create
//    }

//    private void TruckOutput(TruckMaker truck){
//    System.out.println("Truck front variables " + truck.X1 + ", "
+ truck.Y1 + ", " + truck.X2 + ", " + truck.Y2);
//}
      public void individualEvaluate(double x1, double y1, double
x2, double y2){
            TruckMaker truck = new TruckMaker(x1,y1,x2,y2);
            GambitTrucks gt = new GambitTrucks(truck);
            gt.buildTruck(timeStep);
            gt.publishFile("truck.dat");
      }

      public static void main(String[] args){
            GATruck_Miles firstTry = new
GATruck_Miles(10,100,0.5,0.05);
            //TruckMaker truck = firstTry.generateIndividual();
            //firstTry.runOptimization();
            firstTry.individualEvaluate(-1.3412659889425127,-0.222,-
1.7720019384233845,-0.021087335036547118);

//          firstTry.TruckOutput(truck);
      }
}
```

```
package gatruck_Miles;

import java.util.Vector;
```

```
      /*
       * @(#)BubbleSortAlgorithm.java     1.6 95/01/31 James Gosling
       *
       * Copyright (c) 1994 Sun Microsystems, Inc. All Rights
Reserved.
       *
       * Permission to use, copy, modify, and distribute this
software
       * and its documentation for NON-COMMERCIAL purposes and
without
       * fee is hereby granted provided that this copyright notice
       * appears in all copies. Please refer to the file
"copyright.html"
       * for further important copyright and licensing information.
       *
       * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE
SUITABILITY OF
       * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
LIMITED
       * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
       * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE
LIABLE FOR
       * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING,
MODIFYING OR
       * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
       */

      /**
       * A bubble sort demonstration algorithm
       * SortAlgorithm.java, Thu Oct 27 10:32:35 1994
       *
       * @author James Gosling
       * @version       1.6, 31 Jan 1995
       *
       * Modified 23 Jun 1995 by Jason Harrison@cs.ubc.ca:
       *   Algorithm completes early when no items have been swapped
in the
       *   last pass.
       *
       * Modified 18 Dec 2008 by milesbellman@gmail.com
       */
      public class BubbleSorter{
          public static void sort(Generation a){
                  for (int i = a.getGenSize(); --i>=0; ) {
                  boolean flipped = false;
                  for (int j = 0; j<i; j++) {

                        if (a.getTruck(j).getFitness() <
a.getTruck(j+1).getFitness()) {
                              TruckMaker T = a.getTruck(j);
                              a.replaceTruck(a.getTruck(j+1), j);
                              a.replaceTruck(T,j+1);
                              flipped = true;
                        }
                  }
```

```
                if (!flipped) {
                      return;
                }
            }
        }
      }
```

```
/**
 * GambitTrucks.java
 * Author: Miles Bellman
 *
 * This is the class that parameterizes the trucks, sends them to
 * GAMBIT for meshing, and sends the mesh to FLUENT for flow field
 * calculation.
*/

package gatruck_Miles;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;

/**Convert trucks in a generation to coordinates that can be
imported to GAMBIT**/

public class GambitTrucks {
      public static final double x0 = -2;
      public static final double y0 = -0.222;
      public static final double x3 = -0.954;
      public static final double y3 = 0.222;
      public static final double backTopX = 0;
      public static final double backTopY = 0.222;
      public static final double backBottomX = 0;
      public static final double backBottomY = -0.222;
      public static final int extraPoints = 3;
      public double x1,y1,x2,y2,Ax,Bx,Cx,Ay,By,Cy,fitness;
      public double[] gambitTruckXCoordinates,
gambitTruckYCoordinates, cdArray;
      public int iterations;

      public GambitTrucks(TruckMaker truck){
            this.x1 = truck.X1;
            this.y1 = truck.Y1;
            this.x2 = truck.X2;
            this.y2 = truck.Y2;
            this.Cx = getCx(x1,x0);
            this.Bx = getBx(x2,x1,Cx);
            this.Ax = getAx(x3,x0,Cx,Bx);
            this.Cy = getCy(y1,y0);
            this.By = getBy(y2,y1,Cy);
            this.Ay = getAy(y3,y0,Cy,By);
      }

      public void buildTruck(double timeStep){
            iterations = (int) (1/timeStep);
            gambitTruckXCoordinates = new double[iterations +
extraPoints];
            gambitTruckYCoordinates = new double[iterations +
extraPoints];
            double t = 0;

            //bezier curve first
            for(int i=0;i<iterations;i++){
                  gambitTruckXCoordinates[i] = (Ax*Math.pow(t,3)) +
(Bx*Math.pow(t,2)) + (Cx*t) + x0;
                  gambitTruckYCoordinates[i] = (Ay*Math.pow(t,3)) +
(By*Math.pow(t,2)) + (Cy*t) + y0;
                  t = t + timeStep;
            }

            //Add top and back of truck points
            for(int j=iterations;j<iterations+extraPoints;j++){
                  if(j==iterations){
                        gambitTruckXCoordinates[iterations] = x3;
                        gambitTruckYCoordinates[iterations] = y3;
                  } else if(j==iterations+1){
                        gambitTruckXCoordinates[iterations+1] =
backTopX;
                        gambitTruckYCoordinates[iterations+1] =
backTopY;
                  } else{
                        gambitTruckXCoordinates[iterations+2] =
backBottomX;
                        gambitTruckYCoordinates[iterations+2] =
backBottomY;
                  }

            }

            //Scale coordinates so longest part is 1.25m
//          double tip = 0;
//          for(int i=0;i<gambitTruckXCoordinates.length;i++){
//                if(gambitTruckXCoordinates[i] < tip){
//                      tip = gambitTruckXCoordinates[i];
//                }
//          }
//
//          double scaleFactor = -1.25/tip;
//          for(int i=0;i<gambitTruckXCoordinates.length;i++){
//                gambitTruckXCoordinates[i] =
gambitTruckXCoordinates[i] * scaleFactor;
//                gambitTruckYCoordinates[i] =
gambitTruckYCoordinates[i] * scaleFactor;
//          }
```

```
            //Ensure bottom of truck is at 0.05m and back of truck
is at 0.05m
            /*double xDifference = 0.05 -
gambitTruckXCoordinates[gambitTruckXCoordinates.length-1];
            double yDifference = 0.05 -
gambitTruckYCoordinates[gambitTruckYCoordinates.length-1];
            for(int i=0;i<gambitTruckXCoordinates.length;i++){
                  gambitTruckXCoordinates[i] =
gambitTruckXCoordinates[i] + xDifference;
                  gambitTruckYCoordinates[i] =
gambitTruckYCoordinates[i] + yDifference;
            }*/


      }

            /*public void printTruckCoordinates(){
                  for(int i=0;i<truckX.length;i++){
                        System.out.println(truckX[i]+"
"+truckY[i]);
                        System.out.println(truckY[i]);
                        System.out.println("x" + i + ": " +
truckX[i] + "   " + "y" + i + ": " + truckY[i]);
                  }

                  System.out.println("Now Y");

                  for(int i=0;i<truckY.length;i++){
                        System.out.println(truckY[i]+"\n");
                        System.out.println("x" + i + ": " +
truckX[i] + "   " + "y" + i + ": " + truckY[i]);
                  }
            }*/

      public double getFitness(){
            return fitness;
      }

      public double getScoreWithFluent(int generation, int
iteration, double fitness){
            //first check to see if the truck is new or already has
a fitness. If it's new, its fitness will be 100000.
            System.out.println("Fitness: " + fitness);
            if (fitness != 1000000){
                  return fitness;
            }

            else
                  publishFile("truck.dat");
                  //publishFile("truck, " + generation + "-" +
iteration + ".dat");

                  try{
                        Process gambitProc =
Runtime.getRuntime().exec("gambitTest.bat");
```

```
            gambitProc.waitFor();
            Process cleanupProc =
Runtime.getRuntime().exec("cleanup.bat");
            cleanupProc.waitFor();
            Process fluentProc =
Runtime.getRuntime().exec("fluentTest.bat");
            File transcript = new File("trans.jou");
            while(!transcript.exists()){
               //do nothing...we are waiting for
fluent to exit
            }

            BufferedReader cdInput =  new
BufferedReader(new FileReader(new File("cd-history")));
            //BufferedReader clInput =  new
BufferedReader(new FileReader(new File("cl-history")));

            double cd = 1;
            //double cl = -200;
            String line;




            /**Determine fitness from coefficient of
drag history produced by FLUENT**/
            //Skip the first two lines
            for(int i=0;i<2;i++){
               cdInput.readLine();
            }

            //Coefficient of drag is resolved by 19000th
iteration (10000 1st Order, 10000 2nd Order)
            //First: Store last cd iterations in array
for easy evaluation
            int count = 0; //index for cdArray
            cdArray = new double[1000000]; //Must
initialize array with a dimension. This will be more than enough.
            while(cdInput.ready()){
               line = cdInput.readLine(); //Reads the
both the iteration and the cd
               line =
line.substring(line.indexOf("\t")+1); //Makes it so line only reads
the cd
               cdArray[count] =
Double.parseDouble(line); //Stores the cd as a double in cdArray
               count++;
            }

            System.out.println("Count is " + count);
            //Second: Determine if last 1000 cd
iterations are oscillating or flat line
            double sum = 0; //sum of last 1000 cd
iterations
            double avg = 0; //average of last 1000 cd
iterations
```

```
                        for(int i=count-1; i==count-1001; i--){
                              sum = sum + cdArray[i];
                        }
                              //find average of the last 1000
iterations
                              avg = sum/1000;

                              //determine if last 1000 iterations
are flat line and find the flat line
                              if ((avg - cdArray[count-1]) <
0.0001){
                                    cd = cdArray[count-1];
                              }
                              //cd history is oscillating
                              else
                                    cd = avg;

                        /*if(sum != 0)
                              cd = Math.sqrt(sum/((double)(19876 -
19161)));*/

                        /*count = 0;
                        sum = 0;
                        while(clInput.ready()){
                              count++;
                              line = clInput.readLine();
                              if(count > 3500){
                                    line =
line.substring(line.indexOf("\t")+1);
                                    sum += Double.parseDouble(line);
                              }
                        }
                        if(sum != 0)
                              cl = sum/((double)(count - 3500.0));*/

                        /*while(cdInput.ready()){
                              line = cdInput.readLine();
                              line =
line.substring(line.indexOf("\t")+1);
                              cd = Double.parseDouble(line);
                        }*/

                        /*while(clInput.ready()){
                              line = clInput.readLine();
                              line =
line.substring(line.indexOf("\t")+1);
                              cl = Double.parseDouble(line);
                        }*/

                        //System.out.println("Cd = " + cd);
                        cd = Math.abs(cd);
                        if (cd == 0){
                              cd = 10000;
                        }
                        return cd;
                  }
                  catch(Exception e){
                        e.printStackTrace();
                        File transcript = new File("trans.jou");
                        while(!transcript.exists()){
                              //do nothing...we are waiting for
fluent to exit
                        }
                        return 10000;
                  }
      }

      public boolean publishFile(String filename){
              try{
                      // Create file
                      FileWriter fstream = new FileWriter(filename);
                      BufferedWriter out = new
BufferedWriter(fstream);
                      out.write(gambitTruckXCoordinates.length+ "
2\n");
                      for(int
i=0;i<gambitTruckXCoordinates.length;i++){
                        out.write(gambitTruckXCoordinates[i]+ " " +
gambitTruckYCoordinates[i] + " 0\n");
                      }
                      out.close();
                      return true;
              }
              catch (Exception e){
                    System.err.println("Error: " + e.getMessage());
                    return false;
              }
      }
      private double getAx(double x3, double x0, double Cx, double
Bx){
            return x3 - x0 - Cx - Bx;
      }

      private double getBx(double x2, double x1, double Cx){
            return (3*x2) - (3*x1) - Cx;
      }

      private double getCx(double x1, double x0){
            return (3*x1) - (3*x0);
      }

      private double getAy(double y3, double y0, double Cy, double
By){
            return y3 - y0 - Cy - By;
      }

      private double getBy(double y2, double y1, double Cy){
            return (3*y2) - (3*y1) - Cy;
      }

      private double getCy(double y1, double y0){
            return (3*y1) - (3*y0);
```

```
	}

}
```

```
/**
 * Generation.java
 * Author: Miles Bellman
 *
 * This is the class that holds the trucks for evaluation
*/

package gatruck_Miles;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.util.Vector;


public class Generation {
	private Vector<TruckMaker> trucks;
	private int genSize;
	private double timeStep = 0.02;

	public Generation(int genSize){
		this.genSize = genSize;
		this.trucks = new Vector<TruckMaker>();
	}

	/**Add 10 trucks*/
	public void addTruck(TruckMaker truck){
		trucks.add(truck);
	}

	/**Replace a truck at a specific index*/
	public void replaceTruck(TruckMaker truck, int i){
		trucks.removeElementAt(i);
		trucks.insertElementAt(truck,i);
	}

	/**Get a truck*/
	public TruckMaker getTruck(int i){
		return trucks.elementAt(i);
	}

	/**Get truck generation size*/
	public int getGenSize(){
		return genSize;
	}
	/**Get length of vector trucks*/
	public int getTruckVectorSize(){
		return trucks.size();
	}

	/**Remove trucks with "bad" fitness*/
	public void removeTrucks(int num){
		for(int i=0;i<num;i++){
			trucks.removeElementAt(0);
		}
	}

	/**Remove a truck*/
	public boolean removeTruck(TruckMaker truck){
		return trucks.remove(truck);
	}

	/**Get the trucks and their X1,X2 values*/
	public void outputTrucks(){
		for(int i=0;i<trucks.size();i++){
			TruckMaker truck = trucks.elementAt(i);
			System.out.println(truck.X1 + ", " + truck.Y1 + ",
" + truck.X2 + ", " + truck.Y2 + " fitness: " + truck.fitness
+"\n");
		}
	}

	/**create the coordinates for this generation of trucks and
store in vector "truckers"*/
	public void determineFitness(int generation){
		for(int i=0;i<trucks.size();i++){
			TruckMaker truck = trucks.elementAt(i);
			GambitTrucks gt = new GambitTrucks(truck);
			gt.buildTruck(timeStep);
			trucks.elementAt(i).fitness =
gt.getScoreWithFluent(generation,i,trucks.elementAt(i).fitness);
//Math.abs(trucks.elementAt(i).getX1());
			try{
				BufferedWriter recordWriter =  new
BufferedWriter(new FileWriter(new File("record.txt"), true));
				recordWriter.write("Generation: " +
generation + " (" + trucks.elementAt(i).getX1() + "," +
trucks.elementAt(i).getY1() + "," + trucks.elementAt(i).getX2() +
"," + trucks.elementAt(i).getY2() + ") fitness: " +
trucks.elementAt(i).fitness + "\n");
				recordWriter.close();
			}
			catch(Exception e){
				e.printStackTrace();
			}
		}
	}

	/**Get truck of the generation that has the lowest coefficient
of drag*/
	public TruckMaker getBestTruck(){
```

```
            //TruckMaker bestTruck = new TruckMaker();
            for(int i=0;i<trucks.size();i++){
                  //System.out.println("Currently running truck cd = 
" + trucks.elementAt(i).getFitness());
                  if(trucks.elementAt(i).getFitness() < 
GATruck_Miles.bestTruck.getFitness()){
                        GATruck_Miles.bestTruck = 
trucks.elementAt(i);
                        //System.out.println("New best truck cd = " 
+ GATruck_Miles.bestTruck.getFitness());
                  }
                  else{
                        //System.out.println("ELSECurrent best truck 
cd = " + GATruck_Miles.bestTruck.getFitness());
                  }
            }

            return GATruck_Miles.bestTruck;
      }
}
```

```
/**
 * Truck.java
 * Author: Miles Bellman
 *
 * This class ensures the randomly chosen control points are within
 * the specified constraints
*/

package gatruck_Miles;

public class Truck {
      //Bezier boundary coordinates
      public static final double maxX = -0.95;
      public static final double minX = -3;
      public static final double minY = -0.222;
      public static final double maxY = 0.222;

      public static void modTruck(TruckMaker truck){
            //check X0
            /*if (truck.X0 < minX)
                  truck.X0 = minX;
            else if (truck.X0 > maxX)
                  truck.X0 = maxX;
            else
                  truck.X0 = truck.X0;*/

            //check X1
            if (truck.X1 < minX)
                  truck.X1 = minX;
            else if (truck.X1 > maxX)
                  truck.X1 = maxX;
            else
                  truck.X1 = truck.X1;

            //check Y1
            if (truck.Y1 < minY)
                  truck.Y1 = minY;
            else if (truck.Y1 > maxY)
                  truck.Y1 = maxY;
            else
                  truck.Y1 = truck.Y1;

            //check X2
            if (truck.X2 < minX)
                  truck.X2 = minX;
            else if (truck.X2 > maxX)
                  truck.X2 = maxX;
            else
                  truck.X2 = truck.X2;

            //check Y2
            if (truck.Y2 < minY)
                  truck.Y2 = minY;
            else if (truck.Y2 > maxY)
                  truck.Y2 = maxY;
            else
                  truck.Y2 = truck.Y2;

            //check X3
            /*if (truck.X3 < minX)
                  truck.X3 = minX;
            else if (truck.X3 > maxX)
                  truck.X3 = maxX;
            else
                  truck.X3 = truck.X3;*/
      }

}


/**
 * TruckMaker.java
 * Author: Miles Bellman
 *
 * This class creates truck objects
*/

package gatruck_Miles;


public class TruckMaker{
      public double X0,X1,Y1,X2,Y2,X3,fitness;
      public final static double Y0 = -0.222;
      public final static double Y3 = 0.222;
```

```
//    public TruckMaker(TruckMaker truck){
//
      this(truck.X0,truck.X1,truck.Y1,truck.X2,truck.Y2,truck.X3);
//    }
//
//    public TruckMaker(){
//          this(0,0,0,0,0,0);
//    }

      public TruckMaker(double X0, double X1, double Y1, double X2,
double Y2, double X3){
            this.X0 = X0;
            this.X1 = X1;
            this.Y1 = Y1;
            this.X2 = X2;
            this.Y2 = Y2;
            this.fitness = 1000000;
      }

      public TruckMaker(double X1, double Y1, double X2, double Y2){
            this.X1 = X1;
            this.Y1 = Y1;
            this.X2 = X2;
            this.Y2 = Y2;
            this.fitness = 1000000;
      }

      public TruckMaker(double X1, double Y1, double X2, double Y2,
double fitness){
            this.X1 = X1;
            this.Y1 = Y1;
            this.X2 = X2;
            this.Y2 = Y2;
            this.fitness = fitness;
      }
      public TruckMaker(){
            this.fitness = 1000000;
      }

      public double getX0(){
            return X0;
      }
      public double getX1(){
            return X1;
      }
      public double getY1(){
            return Y1;
      }
      public double getX2(){
            return X2;
      }
      public double getY2(){
            return Y2;
      }
      public double getX3(){
            return X3;
```

```
	}
	public double getFitness(){
		return fitness;
	}


}
```

Appendix C

MATLAB postprocessing codes

```
%%Plots the airfoil in the given directory%%

function plotAirfoil(dir,file,shift,color)
[x,y,z] = textread([dir file],'%f%f%f');

%Flip the second half of each vector so the points are in CW order
s = length(x) / 2;
for i = s+1:3/2*s
   temp = x(s*2-(i-(s+1)));
   x(s*2-(i-(s+1))) = x(i);
   x(i) = temp;

   temp = y(s*2-(i-(s+1)));
   y(s*2-(i-(s+1))) = y(i);
   y(i) = temp;
end

plot(x,y+shift,color);
axis equal
```

```
%%Plots the airfoil fitness history for a given GA record%%

clear all

[A B R th fit] = textread('record_sans_ANN.txt','(%f,%f,%f) th: %f
fitness: %f');

dat = zeros(1,length(fit));
dat(1) = fit(1);
for i=2:length(fit)
    dat(i) = max(dat(i-1),fit(i));
end

plot(dat);
xlabel('Number of Evaluations');
ylabel('C_l');
```

```
%%Plots the truck%%
```

```
%%"points" must be specified with the truck coordinates
%%"x_controlPoints" must be specified accordingly

del_t=0.02; % 0.02 is timestep used in java code
N = 1/del_t; % number of iterations

points = [-4.0,0.06696551700106383,-1.2070755950768948,-0.222];

x_controlPoints = [-2,points(1),points(3),-0.954]; % input middle
two manually
y_controlPoints = [-0.222,points(2),points(4),0.222]; % input middle
two manually

%%Bezier curve coefficients
c_x=3*(x_controlPoints(2)-x_controlPoints(1));
b_x=3*(x_controlPoints(3)-x_controlPoints(2))-c_x;
a_x=x_controlPoints(4)-x_controlPoints(1)-c_x-b_x;

c_y=3*(y_controlPoints(2)-y_controlPoints(1));
b_y=3*(y_controlPoints(3)-y_controlPoints(2))-c_y;
a_y=y_controlPoints(4)-y_controlPoints(1)-c_y-b_y;

%%Bezier curve
t=0; X=0; Y=0;
for i=1:N
    X(i)=a_x*(t^3)+b_x*(t^2)+c_x*t+x_controlPoints(1);
    Y(i)=a_y*(t^3)+b_y*(t^2)+c_y*t+y_controlPoints(1);
    t = t + del_t;
end

%% Add other truck points
X(N+1) = -0.954; Y(N+1) = 0.222;
X(N+2) = 0; Y(N+2) = 0.222;
X(N+3) = 0; Y(N+3) = -0.222;
X(N+4) = X(1); Y(N+4) = Y(1);

% %% Scale truck so longest part is 1.25m
% tip = 0;
% for i=1:length(X)
%     if (X(i) < tip)
%         tip = X(i);
%     else
%         % do nothing
%     end
% end
% scaleFactor = -1.25/tip;
% for i=1:length(X)
%     X(i) = X(i) * scaleFactor;
%     Y(i) = Y(i) * scaleFactor;
% end

plot(X,Y); xlabel('x(meters)'); ylabel('y(meters)');
axis([-3 0 -.5 .5]);
%legend('fitness = 1.009');
%%Plots the truck fitness history for a given GA record%%
```

```
clear all

[gen x1 y1 x2 y2 fit] = textread('record.txt','Generation: %f
(%f,%f,%f,%f) fitness: %f');

dat = zeros(1,length(fit));
dat(1) = fit(1);
for i=2:length(fit)
    dat(i) = min(dat(i-1),fit(i));
end

plot(dat);
xlabel('Number of Evaluations');
ylabel('C_d');
```

Appendix D

User Defined Function (UDF) for Synthetic Jet Actuators in FLUENT

```
/*****************************************************************
*******
/* UDF for specifying a transient velocity profile boundary
condition

The only two variables that should change are G, the frequency, and
A, the amplitude   */
/*****************************************************************
******/

#include "udf.h"
#define PI 3.1415926535897932384626433832795

DEFINE_PROFILE(top_inlet_unsteady_velocity, thread, position)
{
  real x[ND_ND];                                  /* holds the
position vector */
  real y, G = 100;
  real z, A = 5;
  float velocity, t;
  face_t f;

  t = RP_Get_Real("flow-time");

   begin_f_loop(f, thread)
      {
         F_CENTROID(x,f,thread);

         F_PROFILE(f, thread, position) = A*(sin(2*PI*G*t));
      }
   end_f_loop(f, thread);
}
```

References

[1] K. Salari, "DOE's Effort to Reduce Truck Aerodynamic Drag Through Joint Experiments and Computations," LLNL-PRES-401649, 28 February 2008.

[2] http://www.livescience.com/environment/etc/090519-obama-announces-new-fuel-efficiency-standards.html

[3] http://www.businessdictionary.com/definition/fuel-efficiency.html

[4] John M. Cimbala and Yunus A. Çengel. *Essentials of Fluid Mechanics: Fundamentals and Applications*. McGraw-Hill, New York, 2008.

[5] A. Seifert, O. Stalnov, D. Sperber, G. Arwatz, V. Palei, S. David, I. Dayan, and I. Fono, "Large Trucks Drag Reduction Using Active Flow Control," AIAA Paper 2008-743, AIAA 46th Aerospace Sciences Meeting and Exhibit, Reno, January 2008.

[6] G. Arwatz, I. Fono, A. Seifert, "Suction and Oscillatory Blowing Actuator," AIAA J. Vol. 45, no. 5, February 2008, pp. 1107.

[7] M. Pastoor, L. Henning, B. R. Noack, R. King, and G. Tadmor, "Feedback Shear Layer Control for Bluff Body Drag Reduction," Journal of Fluid Mechanics, Vol. 608, pp. 161-196, 2008.

[8] A. Brunn, W. Nitsche, L. Henning, and R. King, "Application of Slope-seeking to a Generic Car Model for Active Drag Control," AIAA Paper 2008-6734, AIAA 26th Applied Aerodynamics Conference, Honolulu, August 2008.

[9] S.R. Ahmed, R. Ramm, and G. Faltin, "Some Salient Features of the Time-Averaged Ground Vehicle Wake," SAE-Technical Paper Series, 1984.

[10] E. Wassen and F. Thiele, "Drag Reduction for a Generic Car Model Using Steady Blowing," AIAA Paper 2008-3771, AIAA 4th Flow Control Conference, Seattle, June 2008.

[11] GAMBIT 6.2: Geometry and Mesh Generation Preprocessor, Ansys Inc., 2007.

[12] FLUENT 6.3: Flow Modeling Software, Ansys Inc., 2007.

[13] David E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 1989.

[14] A. Jameson, "Aerodynamic Design via Control Theory," Journal of Scientific Computing, Vol. 3, 1988, pp. 233-260.

[15] B. Morgan, "Gairfoils: Finding High-Lift Joukowski Airfoils with a Genetic Algorithm," Dept. of Mechanical Engineering Technical Report, Washington University in St. Louis, 2007.

[16] I.G. Currie, *Fundamental Mechanics of Fluids*. 3rd edition, Taylor and Francis Group, Boca Raton, 2003.

[17] M. Bellman, B. Morgan, J. Straccia, K. Maschmeyer, and R. K. Agarwal, "Improving Genetic Algorithm Efficiency with an Artificial Neural Network for Optimization of Low Reynolds Number Airfoils," AIAA Paper 2009-1096, 47th AIAA Aerospace Sciences Meeting, Orlando, 2009.

[18] http://www.cfd-online.com/Wiki/Ahmed_body

[19] W. J. Crowther and L. T. Gomes, "An evaluation of the mass and power scaling of synthetic jet actuator flow control technology for civil transport aircraft applications", Proc. IMechE, Vol. 222, Part I: Journal of Systems and Control Engineering, April 2008.

[20] http://www.mcef.ep.usp.br/staff/jmeneg/cesareo/Cesareo_HomePage.html

[21] C. F. Pinzon and R. K. Agarwal, "An Experimental and Computational Study of a Zero-Net-Mass-Flux (ZNMF) Actuator," AIAA Paper 2008-0559, 46th AIAA Aerospace Sciences Meeting, Reno, NV, 7-10 January 2008.

Vita

Miles E. Bellman

Date of Birth	March 15, 1986
Place of Birth	Wayland, Massachusetts
Degrees	Washington University in St. Louis B.S. Cum Laude, Mechanical Engineering December 2008
	Washington University in St. Louis M.S., Mechanical Engineering August 2009
Professional Societies	Sigma Xi, The Scientific Research Society Pi Tau Sigma, The Mechanical Engineering Honor Society
Publications	M. Bellman, B. Morgan, J. Straccia, K. Maschmeyer, and R. K. Agarwal, "Improving Genetic Algorithm Efficiency with an Artificial Neural Network for Optimization of Low Reynolds Number Airfoils," AIAA Paper 2009-1096, 47th AIAA Aerospace Sciences Meeting, Orlando, 2009.

August 2009

Drag Reduction of Generic Truck Models, Bellman, M.S. 2009