Macroscopic Computational Model of Dielectric Barrier Discharge Plasma Actuators

THESIS

Timothy R. Klein, Captain, USAF

AFIT/GAP/ENP/06-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GAP/ENP/06-07

Macroscopic Computational Model of Dielectric Barrier Discharge Plasma Actuators

THESIS

Presented to the Faculty

Department of Engineering Physics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science (Applied Physics)

Timothy R. Klein, BS, EIT

Captain, USAF

February 2006

AFIT/GAP/ENP/06-07

Macroscopic Computational Model of Dielectric Barrier Discharge Plasma Actuators

Timothy R. Klein, BS, EIT

Captain, USAF

Approved:

Wm. F. Bailey	16 Feb '06
Dr. William F. Bailey (Chairman)	date
[illegible]	16 Feb 06
Raymond C. Maple (Member)	date
Richard J. McMullan	16 Feb 06
Richard J. McMullan (Member)	date

AFIT/GAP/ENP/06-07

Abstract

Recent progress in the generation and sustainment of gas discharges at atmospheric pressure has energized research in the field of plasma-aerodynamics. Plasma actuators are promising devices that achieve flow control with no moving parts, do not alter the airfoil shape and place no parts in the flow. The operation of a plasma actuator is examined using a macroscopic (force and power addition) computational fluid dynamic model of a dielectric barrier discharge, DBD, in Fluent®. A parametric approach is adopted to survey the range of requisite magnitudes of momentum and energy delivered to the flow field and to identify the effects of this localized momentum and energy addition on the flow characteristics. Simulations consider the initiation and control of flow over a flat plate in a low velocity fluid. The simulation velocity profiles are compared with the experimental observations of Corke (AIAA 2002-0350) as well as simulations of Font (AIAA 2004-3574), Boeuf and Pitchford (JAP 97 103307 2005), and Roy and Gaitonde (AIAA 2005-4631). The simulation is extended from a flat plate simulation to examine the flow modification over an airfoil. Flow characteristics of lift and drag are compared with experimental results of Post and Corke (AIAA 2003-1024) and the compatible energy/momentum addition is identified. Energy and momentum values are then compared and related to characteristic values arising in DBD operation.

To my wonderful wife and partner in the journey of life.

You are the wind beneath my wings.

Acknowledgements

I express my sincere appreciation to my faculty advisor, Dr. Bailey, for his guidance and support throughout the course of this thesis effort. The insight and experience was certainly appreciated.

I also thank the AFIT Aeronautical Engineering Department for the use of their cluster computers and Fluent® software licenses. Mr. Doak was extremely helpful when I was faced with Unix issues and challenges. LtCol Maple was of vital assistance with grid generation, the beginning uses of Fluent®, and a significant amount of questions concerning fluid dynamics and flow solvers. Maj McMullan was instrumental in setting up my original solver to handle slow flows and turbulence as well as answering my tremendous number of questions concerning fluid dynamics and their solvers.

Timothy R. Klein

Table of Contents

List of Figures

List of Tables

MACROSCOPIC COMPUTATIONAL MODEL
OF DIELECTRIC BARRIER DISCHARGE PLASMA ACTUATORS

I. Introduction

As vehicles are pushed further and further along the envelope of powered flight, certain limits are being reached requiring ingenuity of solution. Specifically, one of these limits is the phenomena of stall on an airfoil or lifting body when it is flown at a high angle of attack. If the airfoil is forced into stall, the condition where lift on the airfoil becomes negligible, the vehicle it is attached to has a tendency to either fall out of the sky or become uncontrollable. Generally, the stall effect occurs when flow over an airfoil becomes separated.

In the past, one solution that was explored was to have vacuums either sucking the flow back to the airfoil or re-energizing the flow by blowing into it. These methods were found to be impractical as debris eventually clogged the tubes.

Another solution, currently in use today, is to use leading edge slats. These devices allow flow from the high pressure lower side of the wing to energize the flow on the upper side of the wing, thus preventing separation. However, these devices cause unwanted vibration and additional drag on the wing. [9]

More recently, experiments have proven that flow can be reattached and controlled using a system of Dielectric Barrier Discharge Plasma Actuators, which will be referred to as DBD's. However, the mechanism affecting the flow is not fully understood. A computational model of the system is needed for optimization of these devices.

The purpose of this research is to computationally simulate, evaluate, and characterize the effects of the addition of momentum and thermal energy, compatible with the operation of a DBD, to the neutral gas flow over a flat plate and an airfoil.

Background

A DBD plasma actuator is defined as “a flow control device with no moving parts, does not change airfoil shape, puts no parts in the flow, and does not suck” [1].

The basic configuration of a DBD plasma actuator is shown in Figure 1. However, there are several different configurations that are possible as seen in Figure 2.

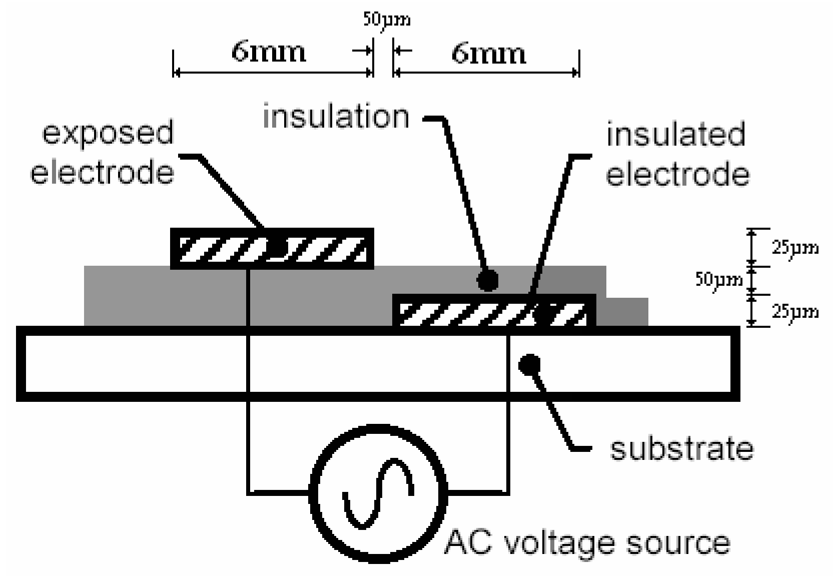


Figure 1
Example of a Dielectric Barrier Discharge (DBD) powered by an AC voltage source. [2]

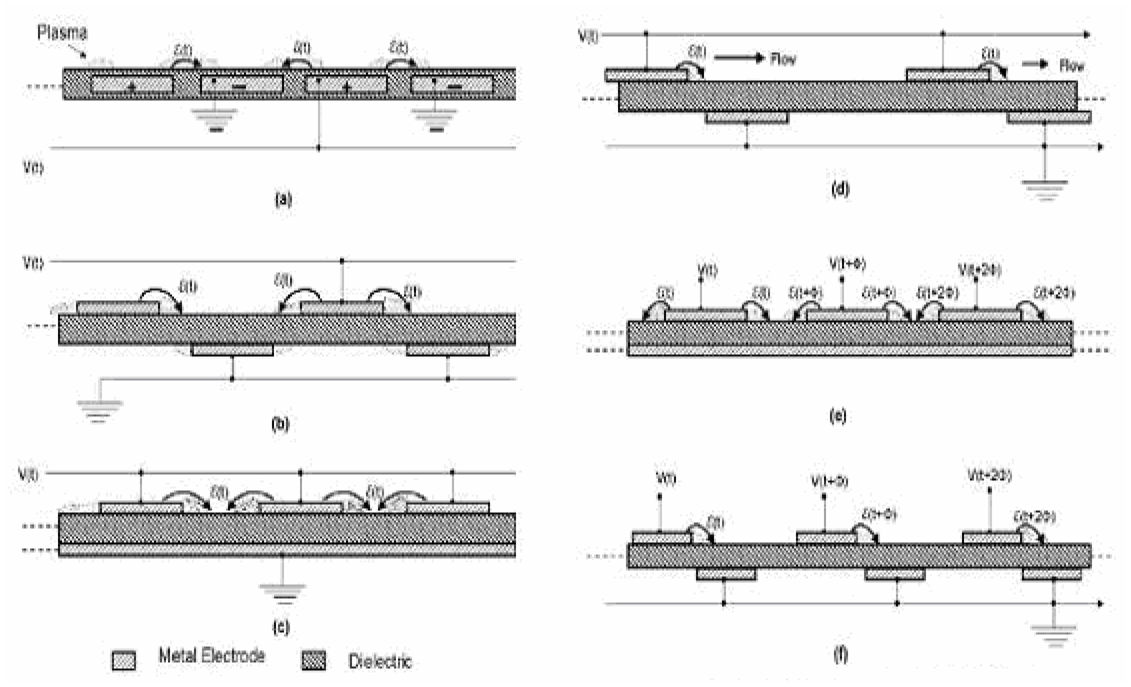


Figure 2
Varying forms of Dielectric Barrier Discharge (DBD) configurations. [3]

Momentum and thermal energy additions transfer forces to the flow through the combinations of collisions of electrons, ionized and neutral particles. The electric field between the two electrodes causes the air to ionize to a quasi-neutral plasma through acceleration of electrons and their subsequent ionization collisions with air molecules. With each collision, given the electron has sufficient energy from its acceleration by the electric field, there is an exchange of thermal energy and a high probability that more electrons will be freed to also be accelerated causing an avalanche effect. This allows the flow to ionize to a quasi-neutral plasma state. The same electric field accelerates the heavy ions in the opposite direction. These ions transfer their momentum to the neutral particles, leading to the modification of the boundary layer flow profile, depicted in Figure 3 and Figure 4.

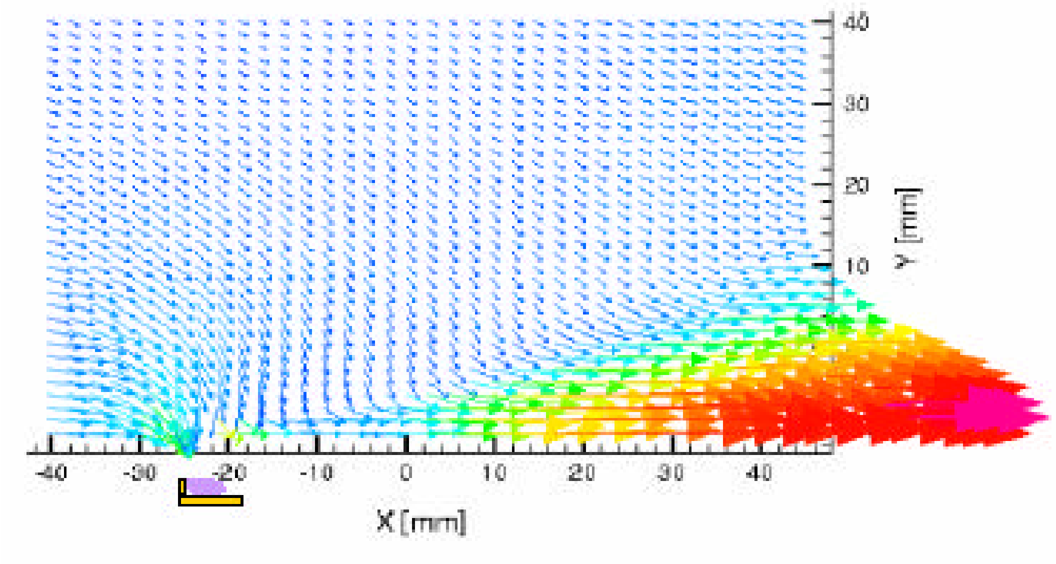


Figure 3
Digital Particle Image Velocimeter (DPIV) Flow Velocity Data
of a DBD in Operation at -25mm [7]

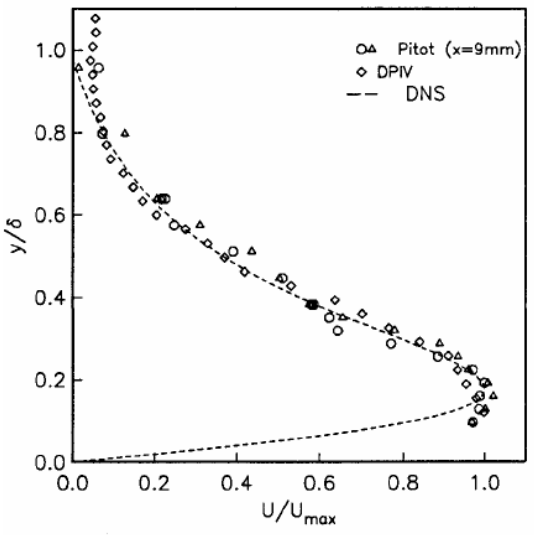


Figure 4
DPIV data of Boundary Layer Flow Velocity;
Normalized by Boundary Layer Thickness and Maximum Velocity [7]

Font [8] describes a single breakdown on each swing of an AC cycle using a PIC code employing a nitrogen chemistry model. Figure 5 displays the electrode configuration of the modeled system. The upper or top electrode is exposed to the flow, while the lower or buried electrode is surrounded by a dielectric material. The simulated spans of the electrodes are 1 cm deep, while the buried electrode is 1.25 mm wide with the exposed electrode 0.25 mm wide.

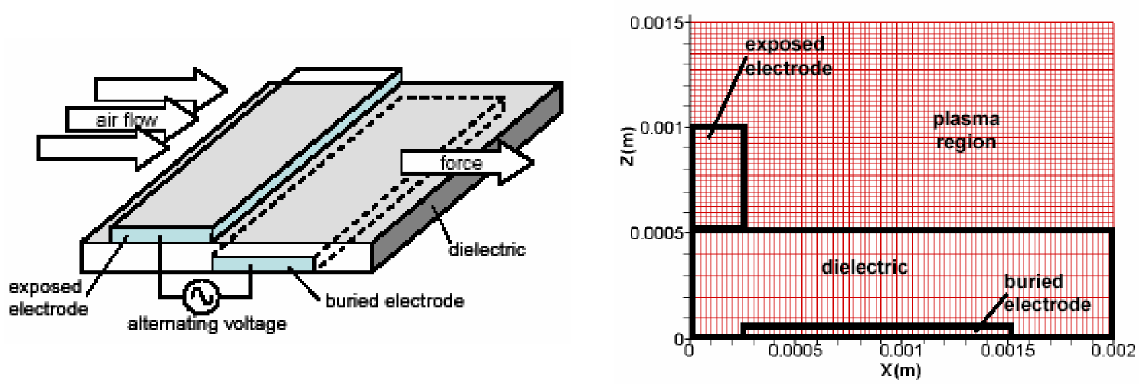


Figure 5
Plasma Actuator Configuration [8]

The results obtained suggest that the majority of ionization occurs on the "backstroke" of the AC cycle, when the upper electrode goes from negative to positive. The data collected is displayed in Figure 6 and Figure 7. The waveform used is a square wave centered about ground, 0V. The first part of the AC cycle is negative, and is referred to as the forward stroke. The second half of the AC cycle is positive, and is referred to as the back stroke. On the forward stroke, the exposed electrode is set to -5000V while the buried electrode is kept at ground, 0V. This has the effect of causing a few random electrons to start a breakdown and sending the free electrons onto the surface of the dielectric barrier. Within 30ns, the electrons accumulate on this surface enough to nullify the field between the two electrodes and the avalanche ceases. During this time, an equal number of ions have also been created and their collisions with neutral particles results in a force $0.2\ m\mathrm{N}$ to the left on the boundary layer flow when this device is operating at 20W. When the back stroke occurs, the exposed electrode is set to $+5000\mathrm{V}$ while the buried electrode is again kept at ground, 0V. The electrons on the surface of the dielectric barrier now accelerate towards the exposed electrode, again causing a breakdown. However, this time there are many more seed electrons, almost all from the surface of the dielectric. Since the electrons were able to nullify the field on the forward stroke, there was -5000V potential at that dielectric location, which results in a total back stroke starting potential of twice the forward stroke $+5\mathrm{kV}-\left(-5\mathrm{kV}\right) = +10\mathrm{kV}$. As a result of more seed electrons and a higher starting potential, a significantly increased amount of ions are produced. This effect can be seen between 60-70ns in Figure 7. Because the upper electrode is exposed, the electrons will impinge upon it and do not nullify the field as they did on the forward stroke. This accounts for the continual increase of ions in Figure 7 on the back stroke. The ions are also pushed away from the exposed electrode and result in a force $1.4\ m\mathrm{N}$ to the right on the boundary layer flow when operating at 20W.

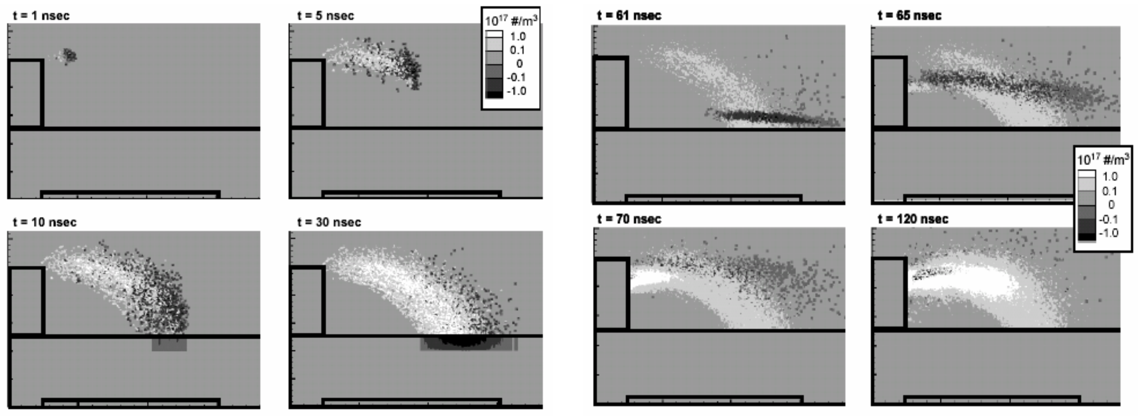


Figure 6
Charge density contours during Forward (t=1-60ns) and Back (t=61-120ns) Strokes [8]
From left to right and top to bottom: t=1ns, 5ns, 61ns, 65ns, 10ns, 30ns, 70ns, 120ns

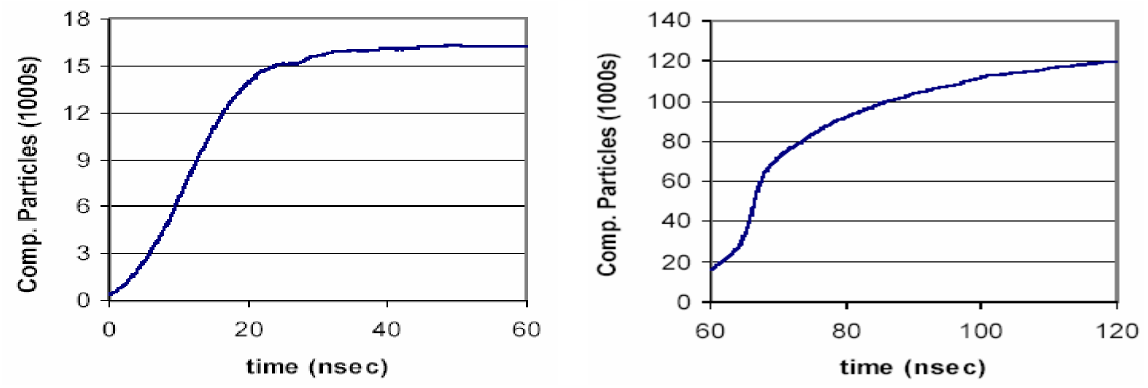


Figure 7
Computed particles during forward stroke (left) and back stroke (right) [8]

Font [8] states that his Particle-In-Cell (PIC) code simulation running at 20 W with a voltage between 1 to 5 kV and frequencies of 1 to 10 kHz will produce "…a net force of $6.0\text{x}10^{-7}$ N". As each cell is $6.25\text{x}10^{-6}$ m^2 with a 0.1mm deep span, we arrive at a unit force per volume of $\frac{6.0\times10^{-7}\,\text{N}}{0.625\times10^{-9}\text{m}^3} = 960\text{N/m}^3 \approx 10^3\,\text{N/m}^3$.

Flows are more likely to become separated if they are laminar than if they are turbulent. This is due to the boundary layer being much larger in a turbulent case resulting in less shear force in the boundary layer. This effect is visualized using smoke flows in Figure 8 and Figure 9. The flow in the top panels of both the curved surface and the sharp corner surface are laminar and both flows are seen to have separated boundary layers near their highest points. In the bottom panel in both figures the flow has been "tripped" to turbulent, thus causing the boundary layer to remain attached for a longer period of time or remain attached for the full length of the figure.

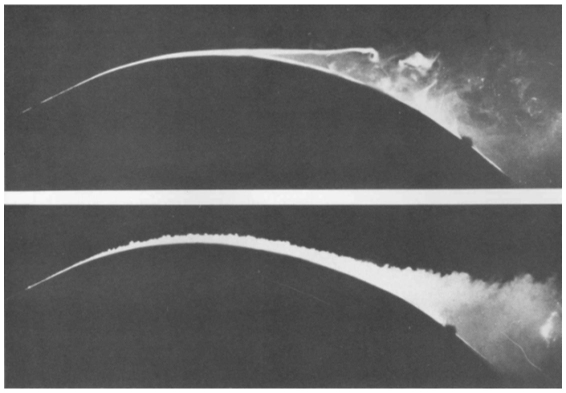


Figure 8
Flow over a Curved Convex Surface;
Laminar (top) and Turbulent (bottom) [6]

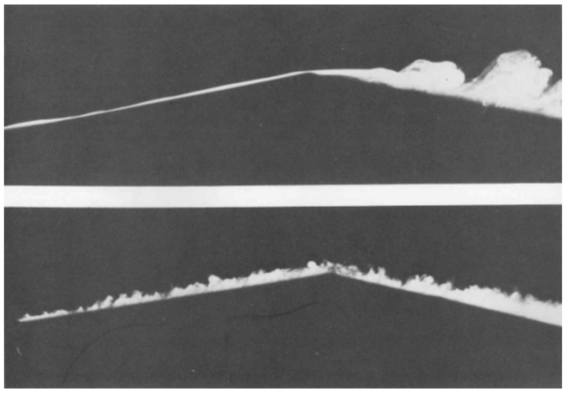


Figure 9
Flow over a Sharp Corner Convex Surface;
Laminar (top) and Turbulent (bottom) [6]

The DBD operation may also "trip" the flow to turbulent earlier, thus maintaining attachment. However, due to the low velocities, hence low Reynolds numbers, that are under examination in this paper, this is not the suspect reason for maintaining attachment in reported experiments [7]. The Reynolds numbers associated with the velocities under examination are $< 0.5 \times 10^6$, which are consistent for laminar flows.

By adding additional momentum and thermal energy via the use of DBD's, the flow is energized and remains attached. Figure 10 and Figure 11 depict the phenomenon of energizing the flow to maintain attachment at large angles of attack where separation is expected. The pictures to the left of each set have the DBD operation set off and show the expected separation. The pictures to the right of each set have the DBD operation set on and show attachment being maintained. The effect of energizing and maintaining attachment of the flow over the airfoil will increase the lift coefficient at a given angle of attach as well as increasing the stall angle for the airfoil.

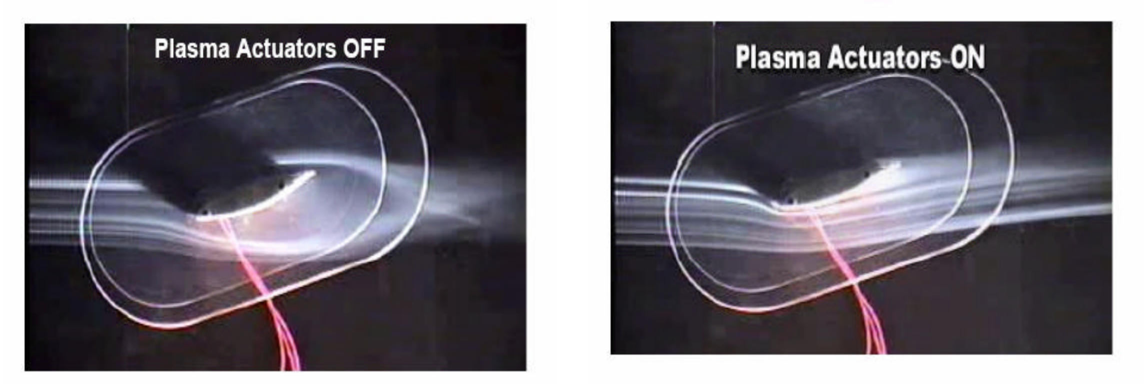


Figure 10
Reattachment of Separated Flow with Actuator ON for
NACA 66_3-018 Airfoil at α=-16° (Smoke used for visualization) [4]

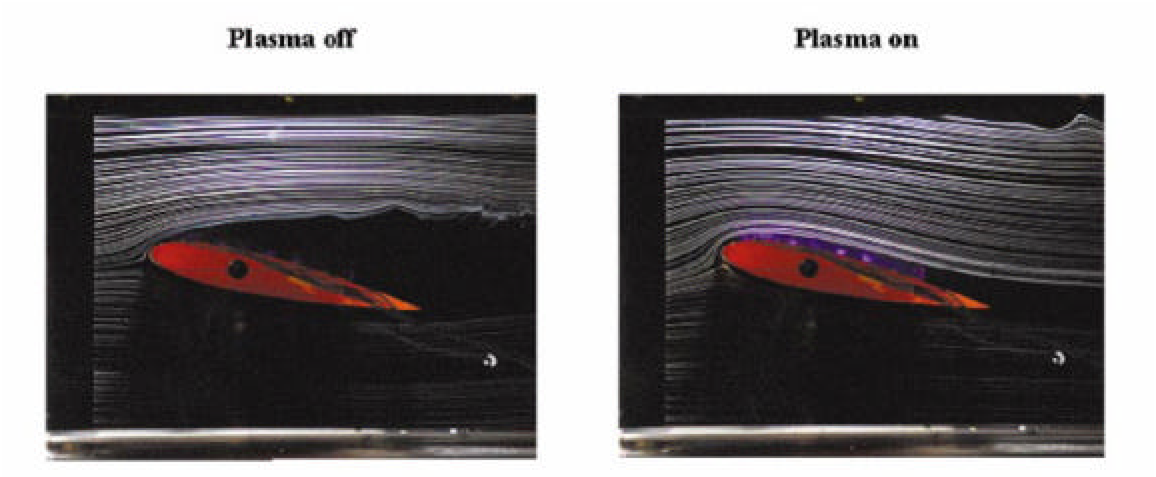


Figure 11
Reattachment of Separated Flow with Actuator ON for
NACA 0015 Airfoil at α=12° (Smoke used for visualization) [5]

Approach

The commercial code, Fluent®, will be used for these simulations and Gridgen® will be used to create the grids.

The research presented requires low velocities at ~2.0 m/s (near-stationary flow) and $\mathrm{v} = U_\infty = \{15.2, 30.4\}$ m/s in order to scope the trade space and compare against experimental data displayed in Figure 12, Figure 13, Figure 14, and Figure 15. These velocities correspond to incompressible flows. A validated incompressible flow solver would take a significant amount of time to create, much more than is reasonable for the purposes of this research effort. This is the main reason for employing Fluent®. Fluent® is a commercial software package that can solve 2-D and 3-D fluid flow simulations. It can handle a wide variety of flow conditions, such as compressible and incompressible flows. An implicit incompressible method of an unsteady time-accurate solution will be used. Within each time step, sub-iterations may be performed to reduce the residual. A maximum of 20 sub-iterations or a tolerance of 10^{-6} for the residual will be used before the solver continues to the next time step. Simulation of the DBD operation will be performed with a set of User Defined Functions (UDF's) implemented in Fluent®.

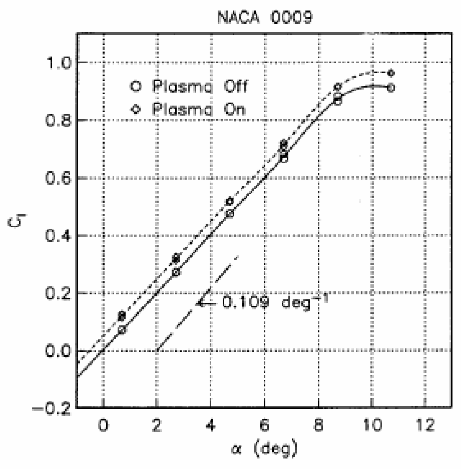


Figure 12
Lift Coefficient vs Angle of Attack for Re_c=180k without and with Actuator Operating [7]

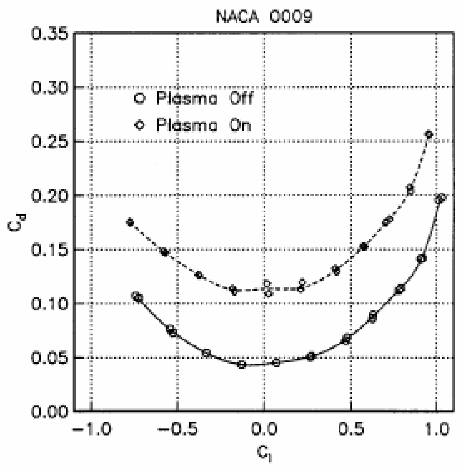


Figure 13
Drag Polar for Re_c=180k without and with Actuator Operating [7]

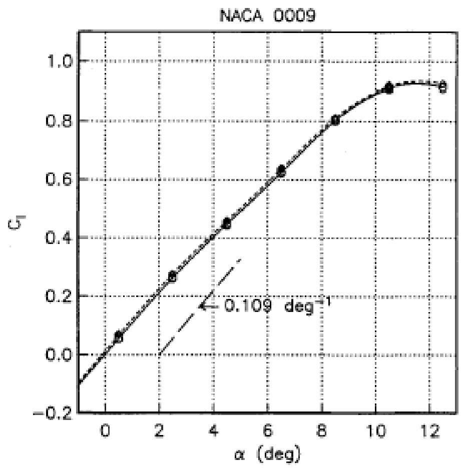


Figure 14
Lift Coefficient vs Angle of Attack for Re_c=360k without and with Actuator Operating [7]

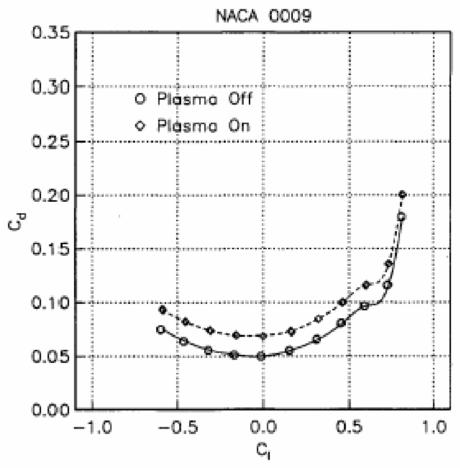
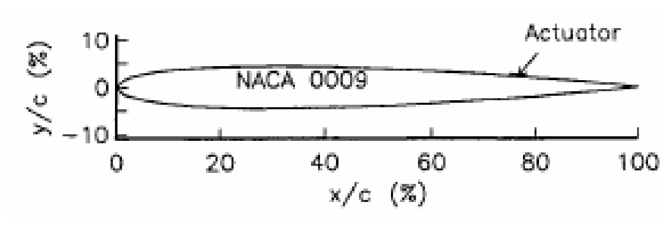
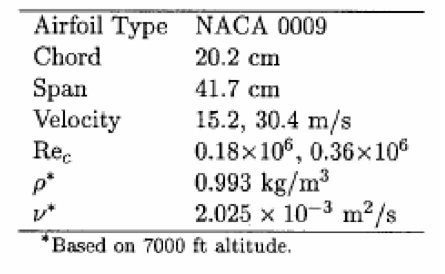


Figure 15
Drag Polar for Re_c=360k without and with Actuator Operating [7]

First, before going into the simulation of a DBD using UDF's, it will be necessary to establish validation of the test cases. The first set of test cases will be a simple flat plate. The second set of test cases will be flow over a NACA 0009 airfoil with the specifications from Figure 16.





Airfoil Type	NACA 0009
Chord	20.2 cm
Span	41.7 cm
Velocity	15.2, 30.4 m/s
Re_c	0.18×10^6, 0.36×10^6
ρ^*	0.993 kg/m^3
ν^*	2.025×10^{-3} m^2/s

*Based on 7000 ft altitude.

Figure 16
NACA0009 Airfoil Test Parameters [7]

The flat plate will be validated by subjecting the grid to both of the above Reynolds' cord numbers for the airfoil, where the Reynolds' number will be assumed to be the same along a flat plate as along an airfoil, $\mathrm{Re}_c = \mathrm{Re}_x = \{0.18\times10^6,\ 0.36\times10^6\}$. The boundary layer profile will be compared to the analytic Blasius differential equations solution for validation.

The airfoil will be validated by subjecting the grid to a set of simulations with $\mathrm{v} = U_\infty = \{15.2,\ 30.4\}$ m/s, corresponding to $\mathrm{Re}_c = \{0.18\times10^6,\ 0.36\times10^6\}$, at angles of attack spanning -16° to $+16^\circ$ in 1° increments. The coefficient of lift, C_L, and the coefficient of drag, C_D, will then be compared to experimentally known data obtained from Selig [15] for validation.

There are many different types of UDF's for Fluent®. Source Term UDF's will be used to calculate the simulated addition of momentum and thermal energy to the flow. A Define on Demand UDF will be used to spatially distribute the momentum and thermal energy addition to the flow. The UDF's are written in the C programming language.

These subroutines will allow a macroscopic simulation of the DBD and are the purpose of this research effort. More detail on these subroutines and their design can be found in the Simulation Setup Section and Appendix B.

Corke et al. [7] present the main mechanism contributing to the boundary layer flow as force created via ion-neutral collisions from the positive ions accelerating in the electric field. A simplified equation for the pressure term coupling to the neutral gas flow is given by Corke et al. [7] in Equation (1).

$$B_E = \tfrac{1}{2} e_0 \nabla E^2 \tag{1}$$

The unknown variable of this equation is the electric field, which includes not only the induced field from the potential between the electrodes, but also includes the field from the plasma as well. As a result, we refer to an updated version of Equation (1) in Equation (2) from Corke et al. [10], which includes the charge density of the plasma estimated from an electrostatic view, but is still an intuitive approximation.

$$\vec{f}_b^{\,*} = r_c \vec{E} = -\left(\frac{e_0}{l_D^{\,2}}\right) j\, \vec{E} \tag{2}$$

In contrast with the previous two equations, Boeuf and Pitchford [16] use an approach relating "…the force per unit volume acting on the gas molecules…" with the number of ions, n_i, the electric field, $\vec{E}$, the ion current density, j_i, and the ion mobility, μ_i, in Equation (3).

$$f \approx e \eta_i \vec{E} = \frac{j_i}{m_i} \tag{3}$$

This equation assumes that the force is primarily transferred in a non-neutral region during an ion and neutral particle collisions where the ion number density is much greater than the electron number density.

The calculations for solving the electric field variable, $\bar{E}$, in Equations (1), (2), and (3) are quite involved and require that the time steps taken be small with respect to the time of one wave cycle. The frequencies used generally reside between 1 kHz and 10 kHz. Frequencies such as these require a significant amount of computational time to arrive at a valid simulation. Therefore, a macroscopic approach of the average effects due to the DBD over several wave cycles is desirable.

The purpose of this research is to examine a macroscopic view (force and power addition) of a DBD in operation. The Source Term UDF will add momentum and energy (time derivatives of force and power respectively) to the flow in an attempt to model the behavior of a DBD without solving a complex and calculation intensive equation. A “weighting” function will assign values to each cell in order to distribute the momentum and energy addition over a particular spatial extent. Further, the time for one period of the AC waveform at a kHz frequency is still much smaller than the anticipated time step associated with the flow simulation. As such, a temporal average of the force over several cycles will be used and implemented via the weighting function. There are currently two different views as to how to spatially distribute the source terms imparted from the DBD on the simulation grid as well as their values.

Boeuf and Pitchford Impulse Density

The first distribution is to employ the source term in an extremely localized set of cells according to Boeuf and Pitchford's impulse simulations [16]. Their simulations were performed with Nitrogen gas at Standard Temperature and Pressure (STP) and account for secondary ionization. The model used does not employ a neutral gas flow solver; instead, it is an ionized gas solver code adapted from their extensive experience with plasma display panels.

The device geometry illustrated in Figure 17 has length scales that are small, 200 μm by 800 μm, with equally small cell sizes (not illustrated in Figure 17) of 2 μm on a side. They state "…the average force per unit volume…will be in the 10^2 - 10^4 N m^{-3} range". The median of this range was estimated to be consistent with the results of Font [8], 10^3 N/m^3.

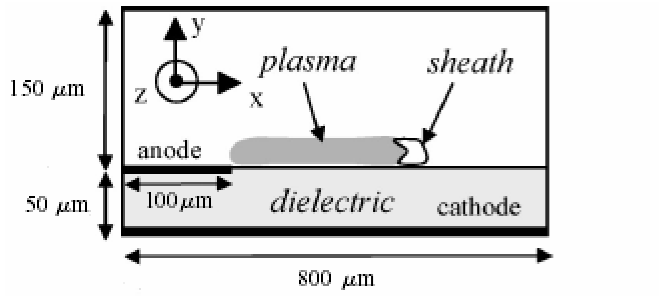


Figure 17
Boeuf and Pitchford Simulation Geometry

Boeuf and Pitchford also report the contours of the impulse density, $F \cdot \Delta t$, around a DBD as shown in Figure 18 and Figure 19. Their simulations used a single square wave. If they used a 1 kHz driving square wave, the impulse density profile would be multiplied by 1000 (=1 kHz) to give a force density.

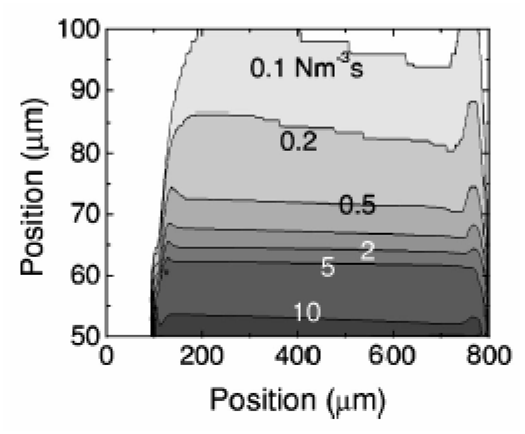


Figure 18
Boeuf and Pitchford [16]
X-Component Impulse Density Weighting

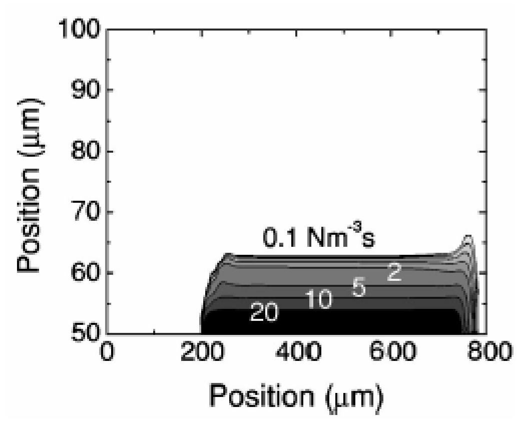


Figure 19
Boeuf and Pitchford [16]
Y-Component Impulse Density Weighting

Finally, Boeuf and Pitchford estimate that the maximum increment to the velocity magnitude of the fluid is directly related to the X-momentum impulse weight by $\Delta v = \frac{1}{r}\int f \ dt$; where the fluid density is $r = 1.2$ kg/m^3, f is the impulse density, and dt is time. Using Figure 18 will produce a wall jet with a velocity between 5 m/s and 10 m/s at a height of 10μm off the surface of the flat plate. The boundary layer profile associated with this effect is depicted in Figure 20 and Figure 21.

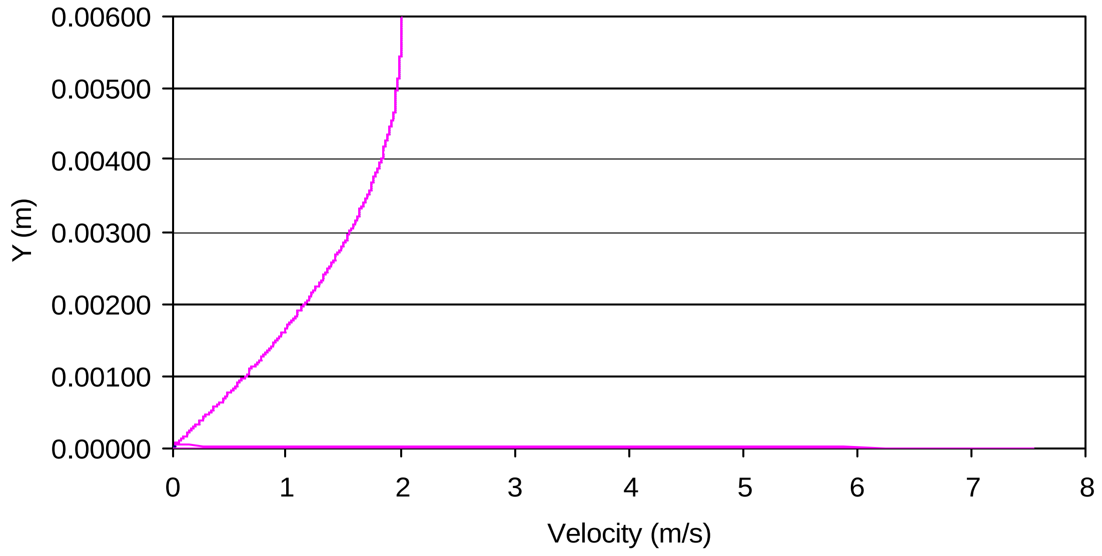


Figure 20
Boeuf and Pitchford Estimation of Wall-Jet Velocity

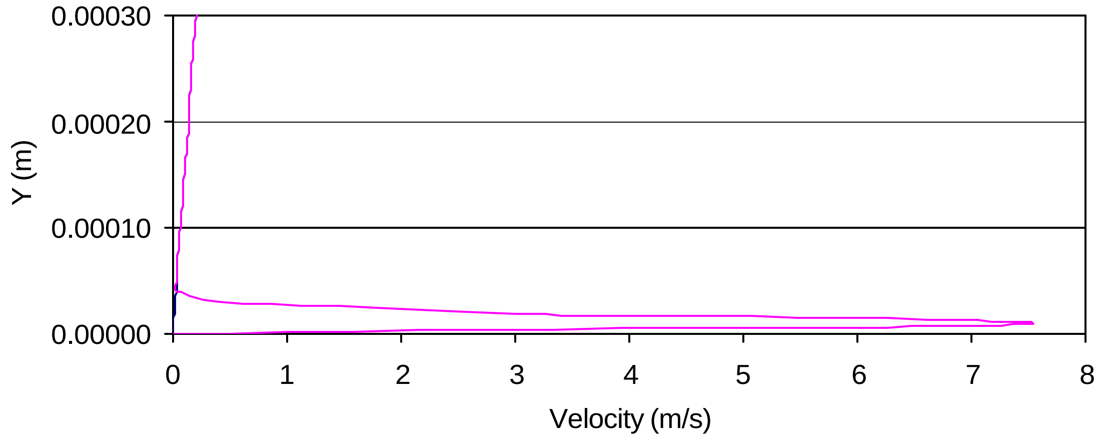


Figure 21
Boeuf and Pitchford Estimation of Wall-Jet Velocity (Close-up)

Roy and Gaitonde Force Density

The second method of distributing the force and thermal addition densities is by Roy and Gaitonde [17], who show that the force density at the DBD is on the order of $10^3\ m\text{N/cm}^3\ \left(10^3\ \text{N/m}^3\right)$. The cm^3 unit volume was confirmed with the authors even though it was not stated specifically in their publication. Further discussions with the authors revealed that the input power to the system per unit length was approximately 7 W/m with negligible amounts of this power going towards thermal heating. Boundary layer velocity profiles reaching 2.5 m/s are shown in Figure 25 and are further discussed in the Results and Conclusions Sections. Their simulations were performed using Helium at STP without secondary emission. The use of Helium instead of a diatomic molecule may have a significant impact on the model's performance and may not mimic atmospheric gas effects correctly as a result.

The Roy and Gaitonde distribution of force density is several orders of magnitude larger than the Boeuf and Pitchford distribution. The dimensions for the computational volume under the DBD influence are 0.5 cm high by 3 cm wide by 1 m deep for the Roy and Gaitonde model, compared to 200 μm (0.02 cm) high by 800 μm (0.08 cm) wide by 1 m deep for the Boeuf and Pitchford model. Further, the widths of the electrodes for each case vary in the same respect. The Roy and Gaitonde geometry uses electrodes that are 1.2 cm wide, while the Boeuf and Pitchford geometry uses electrodes that are 100 μm (0.01 cm) wide for the exposed electrode and an 800 μm (0.08 cm) wide buried electrode that spans the entire simulation space. The disparity in size of the simulation space as well as the electrodes is assumed to have an effect on the performance of the two systems.

Data obtained from Roy and Gaitonde show their simulation results for the induced force densities in Figure 22, Figure 23, and Figure 24. The bold red lines in these figures represent the electrodes of the DBD. As was already reported, these electrodes are 1.2cm wide, two times wider than the electrodes that were used in experiments run by Post and Corke [4] using a NACA 0009 airfoil. Length scales are compared and contrasted later in the validation section. Roy has hypothesized that the long length of the lower electrode allows a charge buildup on the dielectric surface that creates the negative force depicted in Figure 22 and Figure 23. Figure 25 depicts the wall jet velocity profile at 2mm increments, starting from 2mm upstream of the DBD electrode juncture.

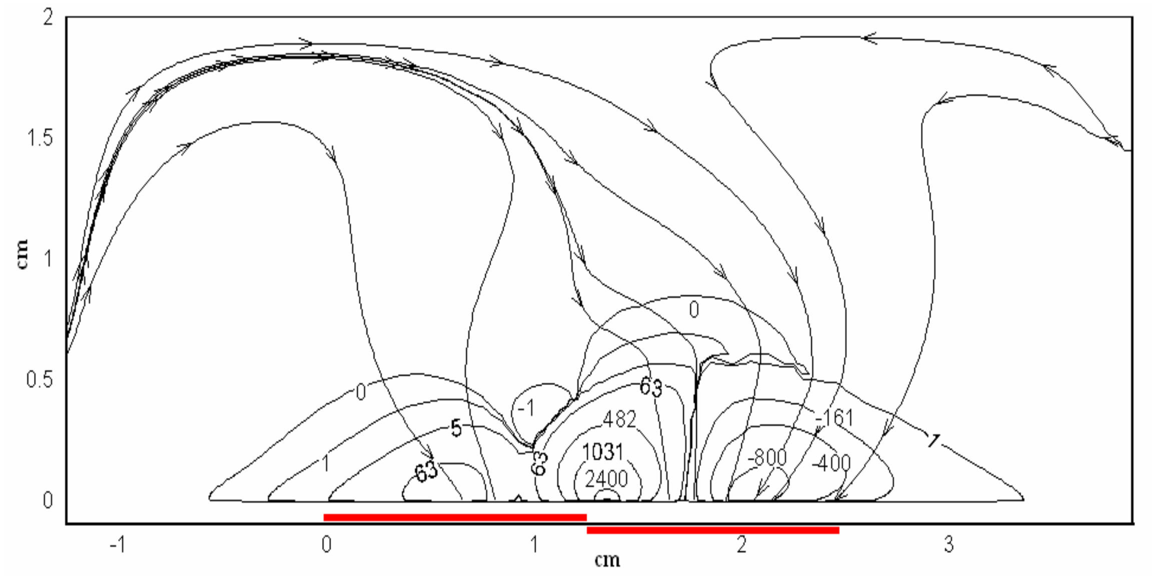


Figure 22
Macroscopic View of X-momentum Force Field (N/m^3) [17]

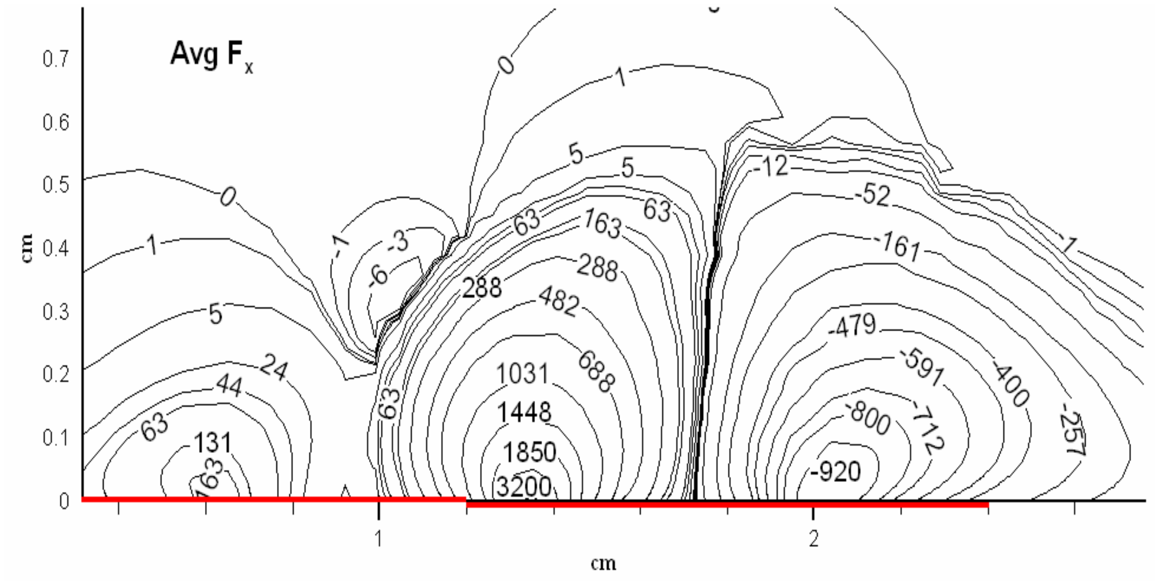


Figure 23
X-momentum Force Field (N/m^3) [17]

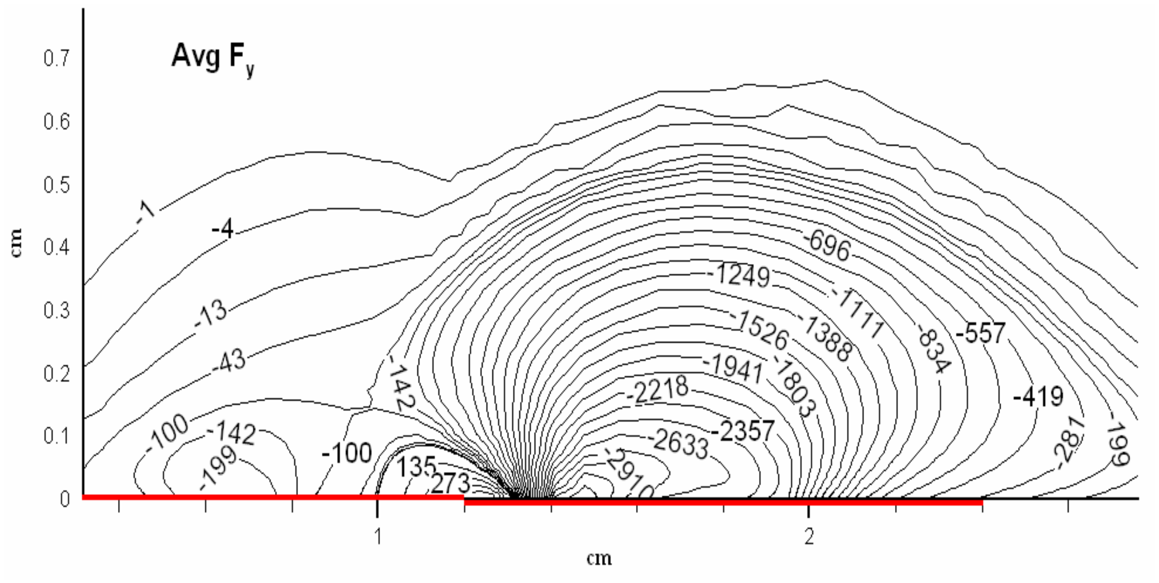


Figure 24
Y-momentum Force Field (N/m^3) [17]

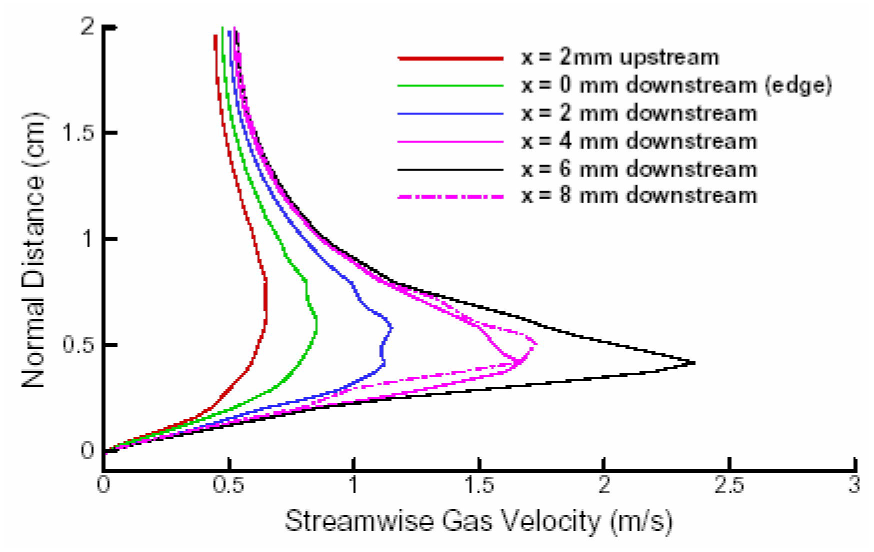


Figure 25
Computed Streamwise Velocity Induced in a Quiescent Helium Gas [17]

From the three sources examined: Font [8], Boeuf and Pitchford [16], and Roy and Gaitonde [17]; the force densities appear to be set around 10^3 N/m^3. Therefore, it is suggested that the momentum source term defined in units of force per unit volume should be near 10^3 N/m^3 to produce a wall jet of ~5 m/s in a near-stationary flow.

Data Set Test Plan

The first set of test cases will simulate flow response using the provided force densities from Boeuf and Pitchford [16] and Roy and Gaitonde [17] on the flat plate grids. As each scheme requires different length scales for the grid cell sizes, two different yet fundamentally similar grids will be required. The geometry of the grids for each of these schemes is explained in detail in the Validation section.

After the initial force density profiles are both complete, several more simulations will be run in order to explore the force density magnitude in each scheme to sufficiently induce a 5 m/s wall jet in a flow of 2 m/s. A simple analysis of heating and momentum addition is performed in the Simulation Setup section. Until this point, little to no thermal energy will have been put into the flow.

Next, thermal energy input will be increased into the flow; showing the effects of thermal energy on the wall jet characteristics. A set of simulations will also be run where the percentage of the components of thermal and momentum addition are kept constant while the total power is increased.

Finally, a NACA 0009 airfoil will be simulated with and without the addition of the DBD simulation source terms. Simulated lift and drag characteristics will be compared to experimental data in Figure 12 thru Figure 15 as reported by Corke [7].

Expectations

This research should establish a macroscopic (force and power addition) computational simulation of a DBD's momentum and thermal energy transfer to a flow. Boundary layer velocity profiles will be obtained and compared with the experimental data reported by Corke [7] and Newcamp [18]. The question of "How much momentum and thermal energy are imparted to the flow and in what fashion?" is to be answered.

II. Simulation Setup

Several pieces of code needed to be developed in order to allow the commercial software, Fluent®, to accurately simulate a DBD. First, there is a subroutine that Fluent® calls once, just prior to starting a simulation. This subroutine stores a "weight" value for each cell in the nearby vicinity of the DBD location so that the three main subroutines can quickly have access to a set of normalized "weighting" data. The three main subroutines are called every time an iteration computes cell data. The subroutines involve the local addition of thermal energy, x-momentum, and y-momentum into the system.

Each time step was 0.001 seconds. This is approximately the amount of time information in the fluid takes to travel the length of the flat plate or airfoil, 0.202 meters, computed by dividing the cord length by the speed of sound. Smaller time steps may increase accuracy of the simulation, but they always increase the computational time. A total of 1000 time steps were completed for a total time of 1 second. Up to 20 sub-iterations are performed for every time step, with convergence of the residual being monitored. If the residual became less than 10^{-6} or all 20 sub-iterations were completed, then the simulation proceeded to the next time step. Monitoring the residual has the practicality for aiding in a more accurate solution using an iterative approach, while still time stepping so as to examine unsteady phenomenon.

Cell Weighting Subroutine

The subroutine that weights cells within the DBD vicinity is extremely important. The spatial extent of the DBD changes depending upon which force density profile is chosen; Boeuf and Pitchford or Roy and Gaitonde. Flexible code is required to adapt to each force density profile and to span over several grid cells, no matter their size and shape. The DEFINE_ON_DEMAND(cell_weight_on_demand) subroutine exercises this function.

The UDF C code is written in several subroutines, all of which are detailed in Appendix B. First, the given location of the DBD is taken from the code and the tangent vector to the surface at that location is found. A coordinate transformation is then employed so that this vector forms the new x-axis. Each transformed point is tested to see if it lies within an estimated influence boundary of the DBD, as predetermined by the force density profile that was chosen for modeling. The power density for thermal addition uses the force density profile to distribute energy, as there is no available data for how the thermal addition is distributed. If the point does lie within an estimated influence boundary, then a series of calculations occur to give the cell a weight. Many issues are taken into account, such as cells straddling one or more boundaries with different equations describing each boundary area. This was one of the most complex parts of this thesis to design. Without it, the developed code would be too rigid for any follow-on work.

The cell weighting equation set can produce force density profiles such as those displayed in Figure 26 and Figure 27. Distances are in meters, with the DBD location set at 75% cord length, or x = 0.1515 meters for a cord length of 0.202 meters. The cell weighting equation set can be found in Appendix B in the Subroutine: weight_funct section. The equations that represent the local weight were created using the force density plots from Roy and Gaitonde and impulse density plots from Boeuf and Pitchford. Equation (4) was used to create a Boeuf and Pitchford force density profile. Similarly, Equation (5) was used to create a Roy and Gaitonde force density profile. In each of these equations, the location of the DBD's center is defined as (x,y) = (0,0). All cells outside of these ranges are given a weight of zero.

$$\left.\begin{array}{c} 10000\exp\left[-50000|x|-80000|y|\right] \\ 10000\exp\left[-100|x|-80000|y|\right] \end{array}\right\} \text{ for } \begin{cases} -0.0001\text{m} \le x \le +0.0000\text{m} \\ +0.0000\text{m} \le x \le +0.0006\text{m} \end{cases} \quad -0.001\text{m} \le y \le +0.00005\text{m} \qquad (4)$$

$$\left.\begin{array}{c} 163\exp\left[-5|x|-5|y|\right] \\ 3200*10^{-275|x|-275|y|} \\ 100*10^{-240|x|-500|y|} \end{array}\right\} \text{ for } \begin{cases} -0.0135\text{m} \le x < -0.0045\text{m} \\ -0.0045\text{m} \le x \le +0.0040\text{m} \\ +0.0040\text{m} < x \le +0.0165\text{m} \end{cases} \quad -0.001\text{m} \le y \le +0.005\text{m} \qquad (5)$$

The values in Equations (4) and (5) were found using a parametric approach to model the behavior of a DBD. The ranges were derived from Figure 18 and Figure 19 for Equation (4), and Figure 23 and Figure 24 for Equation (5). As raw data was not obtained from either Boeuf and Pitchford or Roy and Gaitonde, a direct comparison between the original force density profiles and the fitted force density profile is not available.

The cell weighting code also takes into account the shape of the airfoil at the location of the DBD. This added computational effect can best be seen when comparing the slopes of the gradient contour boundaries between Figure 26 and Figure 27.

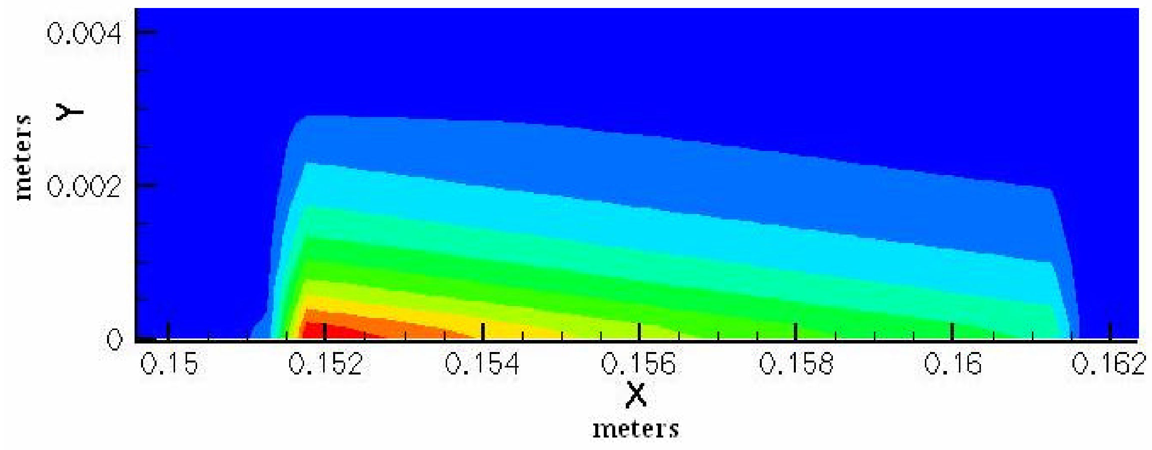


Figure 26
Example of Flat Plate Cell Force Density Profile

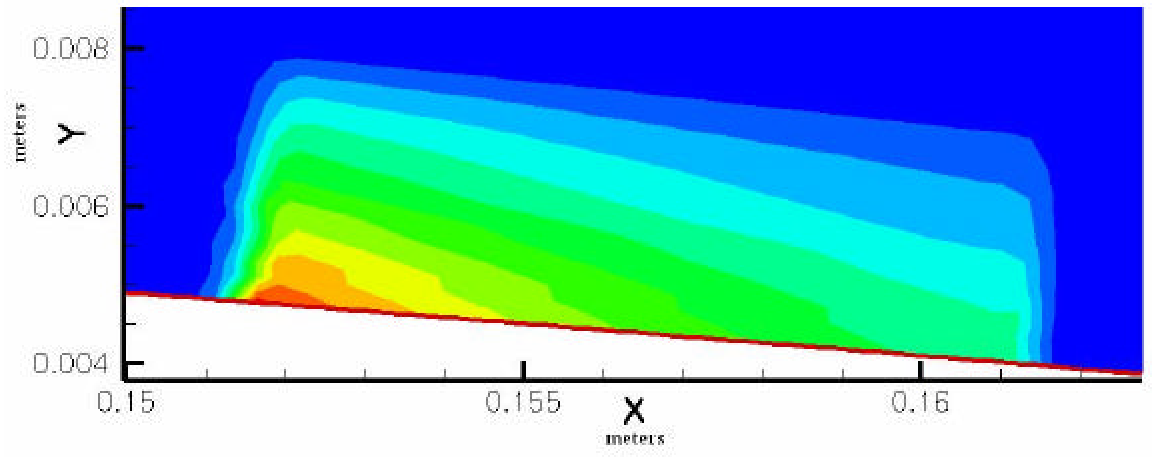


Figure 27
Example of NACA0009 Airfoil Cell Force Density Profile

Once the weight of all of the cells has been determined, the total weight is computed and used to normalize all of the cells' weights. Cells outside the weighted boundary are given a weight of zero. Finally, the cell weight is stored at the corresponding cell's center where Fluent® can call up the data when running the other three main subroutines.

Source Simulation of a DBD

To simulate a DBD, three subroutines were created that add thermal energy (temp_source), x-momentum (x_momentum_source), and y-momentum (y_momentum_source) to the flow. None of these subroutines add mass to the flow, an approach which is typical for a wall jet simulation, but not for a DBD simulation.

The DBD model has to account for certain parameters that are controlled during physical experimentation. The input power in Watts can not be exceeded in the simulation, which is specified in Watts per unit span length of 1 meter for the 2D simulations. Because all three of the source additions must sum to the input power, they must be divided by the following method. A percentage of the input power is given to thermal energy, and the remaining amount necessary to give 100% total power usage was given to momentum, as seen in Equation (6). The power per length in W/m for each of these portions is defined as Thermal Power and Momentum Power, respectively. Each momentum component is then further divided from the power given to momentum, where the total percentage of the x and y components of momentum addition are to equal 100%, as seen in Equation (6). These percentages, multiplied by the cell weight calculated in the cell weighting subroutine, give the total amount of power delivered to a particular cell in terms of power density in units of W/m^3 and directional force density in units of N/m^3. Watts and Newtons are the units of the time derivative for energy and momentum respectively. A derivation for the transformation of power into power density and force density for thermal energy addition and momentum addition, respectively, follows.

$$\begin{gathered}\text{Thermal Energy \% + Momentum \% = 100\%}\\ \text{X-Momentum \% + Y-Momentum \% = 100\%}\end{gathered} \tag{6}$$

$$\begin{gathered}\text{Thermal Energy Power + Momentum Power = Total Input Power}\\ \text{X-Momentum Power + Y-Momentum Power = Momentum Power}\end{gathered} \tag{7}$$

If the thermal energy addition has a considerable effect, then a radial expansion of the flow will be expected where the free stream flow velocity is stationary, as in Figure 3. This is not what is depicted by Digital Particle Image Velocimeter (DPIV) measurements from Figure 3. Instead, the flow is accelerated to the right, which is consistent with momentum addition in the positive x-direction. There is also a component of momentum addition in the y-direction as the flow is drawn towards the DBD.

Extreme care must be taken when implementing and reporting source terms in Fluent®. The documentation and examples supplied by the company and found online are misleading. The units for the thermal energy UDF source and the momentum UDF source may be thought to be J/m^3 and $\mathrm{kg} \cdot (\mathrm{m/s})/\mathrm{m}^3$, respectively. In actuality, the correct units are time derivatives of these units as W/m^3 and N/m^3, respectively.

The correct method for thermal energy addition to the flow is simple. Take the power in Watts to be delivered as thermal energy addition and divide by the cell volume to produce the required input for thermal energy addition with units of W/m^3. Equation (8) shows this in equation form. Due to a lack of experimental and simulated power density distribution data, it was assumed that the power density distribution was similar to the force density distribution. This may or may not be accurate, and an examination of the effect of power density to the system is performed in the Results and Conclusions sections.

$$\text{Power Density}=\frac{\text{Cell\_Weight*Thermal\_Energy\_\%*Total\_Power}}{\text{Cell\_Volume}} \tag{8}$$

A procedure for momentum addition was more difficult to establish. A one dimensional case will be examined for equation development. A two dimensional case would require adding a similar set of equations for the added dimension. Also, because there is a source term subroutine for each momentum addition component, only one velocity component needs to change per momentum subroutine. The kinetic energy of the flow, KE_0, was first solved for using

$$\text{KE}_0 = \tfrac{1}{2} m {u_0}^2 \tag{9}$$

where u_0 is a one dimension velocity component, m is the mass within the cell computed from $m = r \cdot \text{Cell\_Volume}$, and ρ is the fluid density in the cell. Once the fluid's kinetic energy is found, the equation used to derive incremental energy from power is computed in Equation (11). To allow for the flow direction to have effect, the change in kinetic energy, $\triangle\text{KE}$, is defined as positive for forces directed from left to right, and negative for forces directed from right to left. This is due to the cell weight introducing a positive or negative sign as a force density profile requires. The total power is a user input parameter for the simulations and must be positive. In order to check that the program is performing properly, the calculation in Equation (10) may be performed.

$$\text{Total\_Power} = \sum_{1}^{n} \left(\left| \triangle\text{KE}_\text{n} \right| \cdot \triangle\text{t} \right) \tag{10}$$

$$\begin{aligned} &\triangle\text{KE} = \text{Power}*\triangle\text{t} \\ &\text{where }\ \text{Power} = \text{Cell\_Weight*Momentum\_\%*Total\_Power} \end{aligned} \tag{11}$$

The $\triangle$KE is then added to the old kinetic energy, KE_0, to give a new kinetic energy, KE_1.

$$\text{KE}_1 = \tfrac{1}{2}mu_0{}^2 + \triangle\text{KE} \tag{12}$$

Equation (13) is used to preserve direction of a velocity component. The inverse of Equation (13) is Equation (14), where $a = u \cdot |u|$.

$$u^2 \Rightarrow u \cdot |u| \tag{13}$$

$$\begin{aligned} &\text{If } a < 0 \quad u \Rightarrow -\sqrt{|a|} \\ &\text{If } a \geq 0 \quad u \Rightarrow +\sqrt{a} \end{aligned} \tag{14}$$

Briefly, two cases will prove Equation (13) and (14). If $u = 1$, then $u^2 = 1 \cdot |1| = 1$ and the sign is preserved. If $u = (-1)$, then $u^2 = (-1) \cdot |(-1)| = (-1) \cdot |1| = -1$ and the sign is again preserved. Further, if $a = 1$, then $u = +\sqrt{1} = 1$ and the sign is preserved. Similarly, if $a = (-1)$, then $u = -\sqrt{|(-1)|} = -\sqrt{1} = (-1)$ and the sign is again preserved.

The new kinetic energy of the flow can again be described using mass and velocity.

$$\text{KE}_1 = \tfrac{1}{2}mu_1{}^2 \tag{15}$$

Therefore, Equation (13) is substituted into Equation (15) to produce

$$\text{KE}_1 = \tfrac{1}{2}mu_1{}^2 = \tfrac{1}{2}m\left(u_1 \cdot |u_1|\right) \tag{16}$$

Solving for the velocity component yields

$$u_1 \cdot |u_1| = \frac{2 \cdot \text{KE}_1}{m} \tag{17}$$

Substituting in Equation (12) for KE_1

$$u_1 \cdot |u_1| = \frac{2 \cdot \left(\frac{1}{2} m \left(u_0 \cdot |u_0|\right) + \triangle \mathrm{KE}\right)}{m} \tag{18}$$

Simplifying produces

$$u_1 \cdot |u_1| = \left(u_0 \cdot |u_0|\right) + \frac{2 \cdot \triangle \mathrm{KE}}{m} \tag{19}$$

If the quantity in Equation (19) is negative, then an additional absolute value and negative sign must be implemented to keep the velocity sign consistent as was shown with Equation (14). This is shown with Equation (20). If the new flow velocity is positive, or flows to the right, then Equation (21) would be used.

$$u_1 = -\sqrt{\left|\left(u_0 \cdot |u_0|\right) + \frac{2 \cdot (\mathrm{Cell\_Weight*Momentum\_\%*Total\_Power})}{r \cdot \mathrm{Cell\_Volume}}\right|} \tag{20}$$

$$u_1 = +\sqrt{\left(u_0 \cdot |u_0|\right) + \frac{2 \cdot (\mathrm{Cell\_Weight*Momentum\_\%*Total\_Power})}{r \cdot \mathrm{Cell\_Volume}}} \tag{21}$$

The difference between the new and the old velocity is then divided by the time step to yield acceleration.

$$a = \frac{u_1 - u_0}{\triangle t} \tag{22}$$

The acceleration multiplied by density gives the source addition term in units of N/m^3. Fluent® requires that a source be in terms of a change per volume, which yields

$$\frac{m \cdot a}{vol} = r \cdot a = r \cdot \frac{u_1 - u_0}{\triangle t} \tag{23}$$

Equation (24) shows this entire procedure in one form with the assumption that the kinetic energy increase, $\vartriangle\text{KE}$, and the initial starting velocity are ≥ 0.

$$\text{Force Density} = r\,\frac{\left(\sqrt{{u_0}^2 + \dfrac{2\cdot\left(\text{Cell\_Weight*Momentum\_\%*Total\_Power}\right)\cdot\Delta\text{t}}{r\cdot\text{Cell\_Volume}}}\right) - {u_0}^2}{\Delta\text{t}} \tag{24}$$

As a check for the Roy and Gaitonde force density profile, the velocity increase was calculated using a time step of 0.001 seconds and 2% momentum from 5 W/m, or 0.10 W/m; a setting that yields a force density of 1800 N/m^3 using Equation (24) with a stationary flow, $u_0 = 0$ m/s. This is consistent with the Roy and Gaitonde force density profile. To compute this increase, the volume was computed from the dimensions of the weighted area of the model. The dimensions are $0.5\text{cm}\times 1.0\text{cm}\times 100\text{cm}\left(1\text{ meter}\right)$, to give a volume of $50\times10^{-6}\text{m}^3 = 50\text{cm}^3$. The density at STP is $r = 1.225$ kg/m^3, and the mass in the volume is found by $m = r\cdot vol$. Assuming a stationary flow, $u_0 = 0$ m/s, using Equation (19) and the positive part of Equation (14), the result yields an estimated increase of 1.8 m/s.

$$u = \sqrt{\frac{2\cdot\vartriangle\text{KE}}{m}} = \sqrt{\frac{2\cdot\text{Pwr}\cdot\Delta\text{t}}{r\cdot vol}} \tag{25}$$

As a check for the Boeuf and Pitchford force density profile, the velocity increase was calculated using a time step of 0.001 seconds and 0.00027 W/m; a setting that will later prove in the Results section to yield a force density consistent with the Boeuf and Pitchford force density profile. The dimensions for the Boeuf and Pitchford geometry are $150m\text{m}\times 800m\text{m}\times 100\text{cm}\left(1\text{ meter}\right)$, to give a volume of $120\times10^{-9}\text{m}^3 = 120\text{mm}^3$. The same density, stationary flow equation, and Equations (24) and (25) were used to yield a force density of 1900 N/m^3 and giving a result of 1.9 m/s for the wall jet velocity, respectively.

Further, when calculating using a moving fluid the equation requires the additional variable of initial flow velocity, u_0. Using the same assumptions of positive flow velocity and positive $\vartriangle$KE the equation now becomes

$$u = \sqrt{u_0{}^2 + \frac{2 \cdot \vartriangle \mathrm{KE}}{m}} = \sqrt{u_0{}^2 + \frac{2 \cdot \mathrm{Pwr} \cdot \Delta \mathrm{t}}{r \cdot vol}} \tag{26}$$

If the increment of energy to the cell is fixed by a force density profile, then a relationship between the new velocity and the initial may be graphed, as is done in Figure 28 and Figure 29.

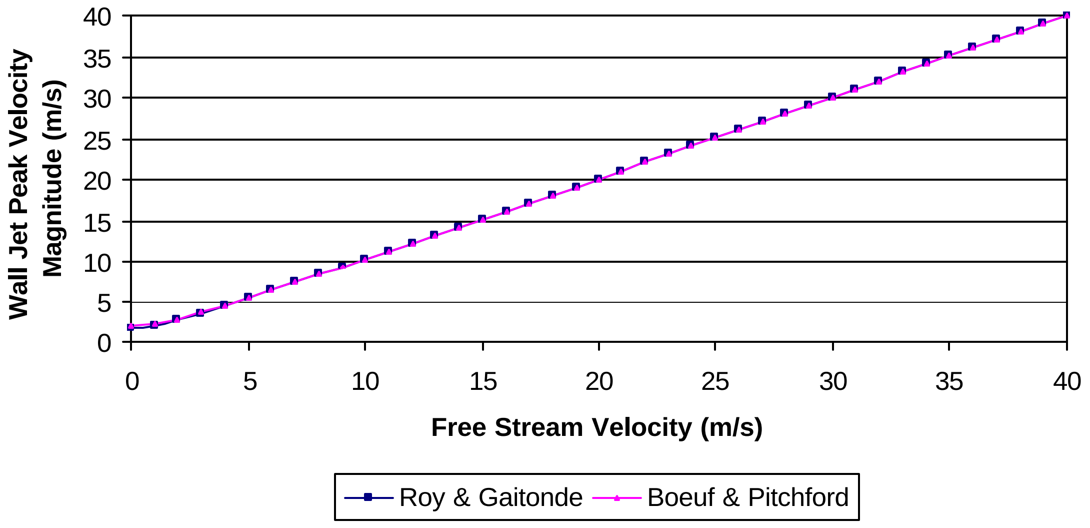


Figure 28
Estimated Wall Jet Peak Velocity Magnitude (m/s)
Compared to the Free Stream Velocity (m/s)

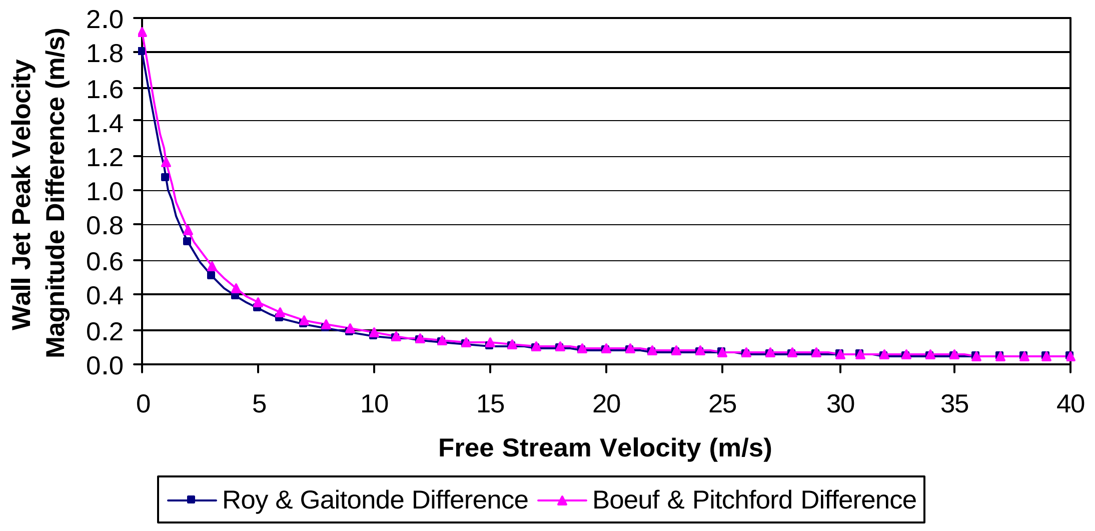


Figure 29
Difference between the Estimated Wall Jet Peak Velocity Magnitude
and the Free Stream Velocity (m/s)
Compared to the Free Stream Velocity (m/s)

As with the experiments, as the free stream velocity is increased, the wall jet becomes less notable. This is because the increment in energy delivered to the flow remains the same, but the fluid's kinetic energy increases as the free stream velocity increases. Eventually, the energy increment becomes insignificant when compared to the kinetic energy of the fast moving free stream velocity and little to no change in the velocity profile is seen.

Now that the momentum and energy addition subroutines have been explained and their effects on the flow estimated, it is necessary to describe the term ds[eqn]. In all three of the subroutines, the term ds[eqn] is set to 0.0. This term is the derivative of the source term if known or 0.0 if unknown. Because the derivatives of the source terms are complex and dependent upon the model, Fluent® was employed to explicitly solve for the derivative in its attempt to derive a better and more stable solution for its implicit solver. A sample momentum source term that the Fluent® manual supplied was run with ds[eqn] set equal to the derivative of a source and ds[eqn]=0.0 in a second case. Both cases converged in the same number of iterations and had the exact same results. The clock time difference was measured as insignificant for this test run set and is not expected to have any significant impact on simulation run times.

Finally, to ensure that the additional subroutines written were all functioning correctly, a set of verification runs outside of Fluent® were performed. Further detail of each subroutine and its workings are discussed in Appendix B. The verification for each subroutine is also discussed there as well.

III. Validation

Calculation for Boundary Layer Thickness and Flat Plate Grid Geometry

For flow along a flat plate as seen in Figure 30, the boundary conditions are no-slip at the wall, $u(0)=0$, and a smooth transition of the flow to the free stream velocity, $u(d_{99})=U_\infty$ and $\left.\frac{\partial u}{\partial y}\right|_{y=d_{99}}=0$. If the flow is laminar, these boundary layer velocities can be approximated with a second order polynomial approximation in Equation (27), given by White [11:222], equation 4-11. The Blasius formula for determining where the boundary layer thickness is 99% of the free stream flow is given in Equation (28), from White [11:223], equation 4-14; where ρ is density with units of kg/m^3, $\mathrm{v}=U_\infty$ is the free stream velocity with units of m/s, μ is viscosity with units of Ns/m^2, and x is the distance in meters from the leading edge of the flat plate. Equation (29) is the Reynold's number at a point x along the flat plate from the leading edge.

$$u(y)\approx U_\infty\left(\frac{2y}{d_{99}}-\frac{y^2}{d_{99}{}^2}\right) \tag{27}$$

$$d_{99}\approx\frac{5.5x}{\sqrt{\mathrm{Re}_x}} \tag{28}$$

$$\mathrm{Re}_x=\frac{r\cdot\mathrm{v}}{m}x \tag{29}$$

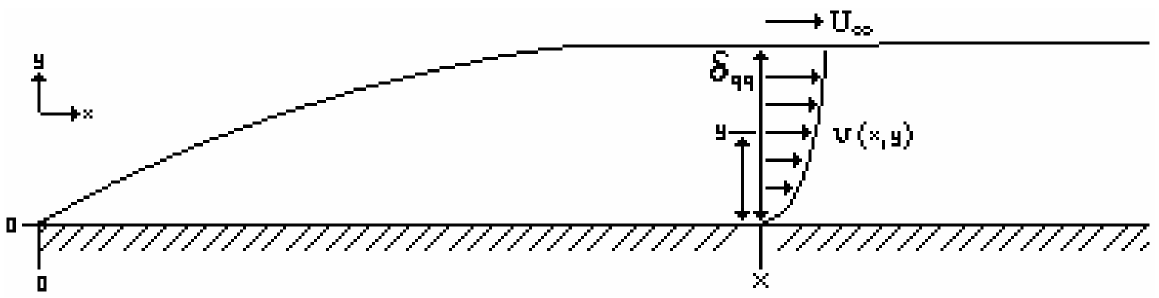


Figure 30
Boundary Layer Velocity Profile for a Flat Plate

The goal for validation is to model the DBD operation at near stationary flow velocities at STP and with the same setup as the NACA0009 airfoil experiments. From the table in Figure 16, we can extract $r = 0.993 \text{ kg/m}^3$, $\text{v} = U_\infty = \{15.2, 30.4\} \text{ m/s}$, the cord length $c = 0.202\text{m}$, and $\text{Re}_c = \{0.18\times10^6, 0.36\times10^6\}$ for the NACA0009 airfoil experiments reported by Corke [7]. Because the viscosity at 7000 feet altitude was not given, the Re_x equation was inverted for μ, giving $m = 1.69\times10^{-5} \text{ N}\cdot\text{s/m}^2$. This corresponds to kinematic viscosity, $u = \frac{m}{r} = 1.70\times10^{-5} \text{ m}^2\text{/s}$. The temperature was not published and will be assumed to be 288.15 °K, with a pressure of 78669 Pa based on a 7000 foot (2133.5 meters) altitude, derived from the equation $p = p_0 \exp\left[\frac{-g}{R*T} z\right]$; where $p_0 = 101325 \text{ Pa}$, $R = 287 \text{ J/kg}^\circ\text{K}$, $g = 9.81 \text{ m/s}^2$, and $T = 288.15^\circ\text{K}$.

The flat plate was run at the same Reynolds' numbers as the airfoil. Therefore, the length of the plate will be set to 0.202 meters to accommodate a similar velocity, density, and viscosity as the airfoil experiment. For verification that the viscosity is in the correct range, the values for air at STP are referenced in Table 1.

Table 1. Standard Temperature and Pressure (STP) for Air

Temperature:	288.15 °K
Pressure:	101325 Pa
Speed of Sound:	340.2 m/s
μ:	$1.7894\times10^{-5}\,\mathrm{N\cdot s/m^2}$
ρ:	$1.225\ \mathrm{kg/m^3}$

In White [11], it is found that these flows will be laminar, not turbulent, in nature when comparing the Reynolds numbers to the statement "the boundary-layer flow is likely to be laminar in the range $1000 < Re < 10^6$" [11:218].

For the two flow speeds, the maximum and minimum boundary layer thickness was used to facilitate creation of a flat plate grid. The value of x was set to 75% of the length of the plate, $x = 0.1515\,\mathrm{m}$, to ensure that the boundary layer is fully developed and to simulate where the DBD will be placed along the airfoil as seen in Figure 16. For further validation and calculations, a simulation of the boundary layer velocity profile was performed for a value of x set to near 100% of the length of the plate, $x = 0.2015\,\mathrm{m}$. A simulation at this point allowed for easy verification calculations with the Fluent® simulations and for determining the maximum height needed for the grid spacing.

To examine how the boundary layer will behave along the flat plate at different velocities, the following equation sets in Table 2 as well as Figure 31 and Figure 32 were created. The graphs and equation sets depict a boundary layer velocity profile at a position 75% and near 100% down the length of the plate as already described. The analytic equations used are the Blasius boundary layer solutions to the differential flow equations.

Table 2. Blasius Boundary Layer Velocity Profile Calculations

$\text{v} = U_\infty = 2$ m/s $r = 1.225\ \text{kg/m}^3 \quad m = 1.79 \cdot 10^{-5}\,\text{N}\cdot\text{s/m}^2 \quad u = \frac{m}{r} = 1.46\times10^{-5}\,\text{m}^2\text{/s}$	
$x = (75\%)(0.202) = 0.1515\,\text{meters}$	$x = 0.2015\,\text{meters}$
$\text{Re}_x = \frac{r\cdot \text{v}}{m}x = \frac{\text{v}}{u}x = 0.021\times10^6$	$\text{Re}_x = 0.027\times10^6$
$d_{99} \approx \frac{5.5x}{\sqrt{\text{Re}_x}} = 5.79\times10^{-3}\,\text{meters} = 5.79\ \text{mm}$	$d_{99} \approx \frac{5.5x}{\sqrt{\text{Re}_x}} = 6.67\times10^{-3}\,\text{meters} = 6.67\ \text{mm}$
$u \approx U_\infty\left(\frac{2y}{d_{99}} - \frac{y^2}{d_{99}{}^2}\right) = 691y - 59700y^2$	$u \approx U_\infty\left(\frac{2y}{d_{99}} - \frac{y^2}{d_{99}{}^2}\right) = 599y - 44900y^2$

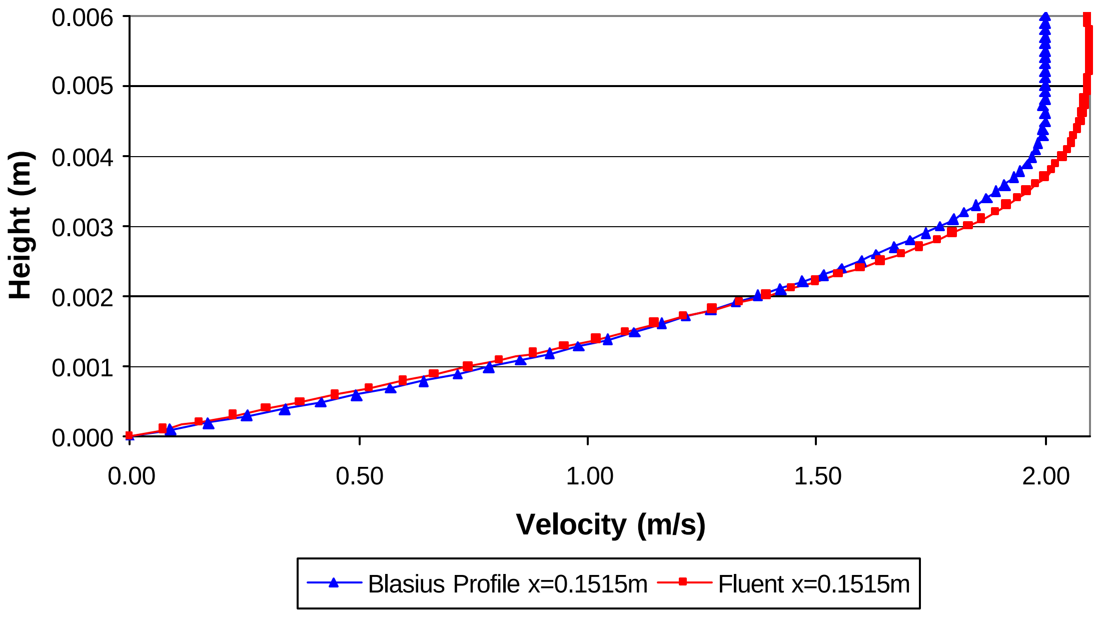


Figure 31
2 m/s Freestream Velocity Profile at x=0.1515 meters
Analytic Blasius Velocity Profile VS Fluent Data

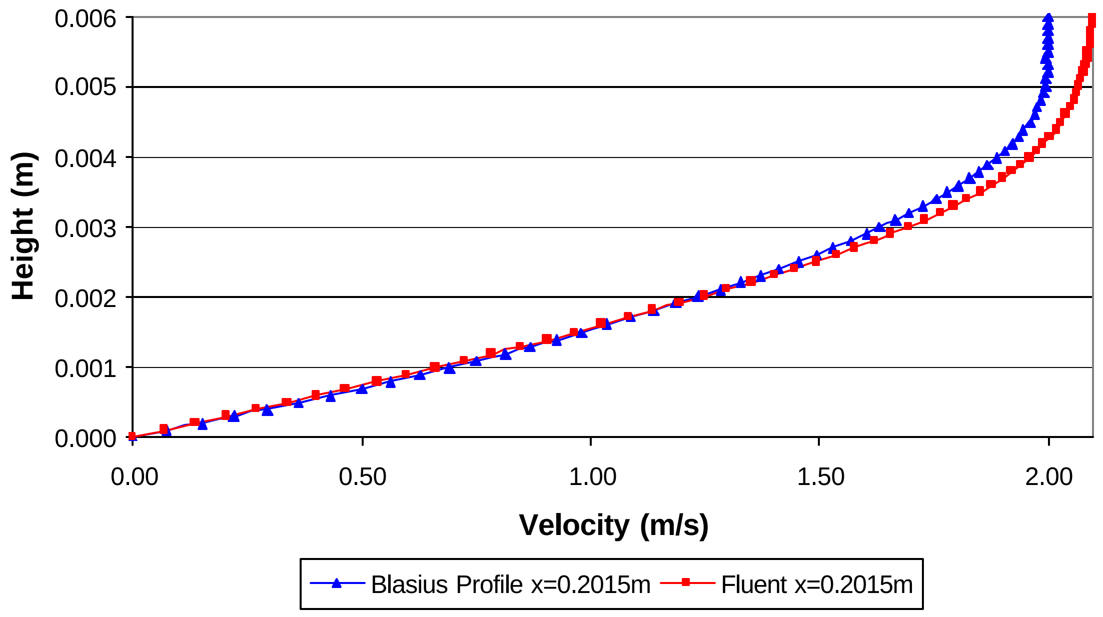


Figure 32
2 m/s Freestream Velocity Profile at x=0.2015 meters
Analytic Blasius Velocity Profile VS Fluent Data

Horizontal Grid Spacing

The flat plate grid is the main test platform and was modeled prior to the airfoil grid creation. The flat plate used a total of 250 cells from 0 to 0.202 meters with a tanh(x) spacing and initial spacing of 1×10^{-4} at the leading edge. A total of 96 points from $0 \le x \le 0.55c$ $(0 \le x \le 0.111)$ meters was set; where the cord length is defined as c=0.202m from the table in Figure 16. The DPIV data depicted in Figure 3 was used as a guide for increased cell density as it shows significant wall jet induced velocities from $(-0.020 \le d \le +0.060)$ meters, where $d = 0.75c$ $(d = 0.1515\text{ m})$ from Figure 16. Finally, small cell sizes yielding more fidelity but more calculation time and possible instabilities were used in the DBD profile region.

The largest cell in the region of the DBD depends upon the force density scheme applied. Thus, a grid for the Boeuf and Pitchford and a separate grid for the Roy and Gaitonde profiles were created. The Boeuf and Pitchford profile is so incredibly small compared to the entirety of the grid that 101 points were compressed into 1mm between $\left(0.151\text{m} \le x \le 0.152\text{m}\right)$. The even spacing of these points provides a distance of 10 μm per side of the cell. The Roy and Gaitonde profile is much larger, and therefore has a span of 39mm between $\left(0.141\text{m} \le x \le 0.180\text{m}\right)$. The even spacing of 157 points yields a distance of 250 μm per side of the cell.

Vertical Grid Spacing

The maximum height of the flat plate grid needs to be calculated using the thickest boundary layer expected, which is obtained when the free stream velocity is slowest. The maximum height is set greater than or equal to $4 \cdot \left(d_{99_{\text{SlowBL}}}\right)$, where $d_{99_{\text{SlowBL}}}$ was originally set for a 15.2 m/s scenario; this yielded $4 \cdot \left(2.62 \times 10^{-3}\right)$ meters, or 10.2mm. The height of the smallest cell needs to be calculated in just the opposite manner, when the thinnest boundary layer is expected, which is when the free stream velocity is fastest. The minimum height is set to less than or equal to $\frac{d_{99_{\text{FastBL}}}}{10}$ in order to have about 10 cells in the y-direction to capture the boundary layer, where $d_{99_{\text{FastBL}}}$ was originally set for a 30.4 m/s scenario; this yielded $\frac{1.60 \times 10^{-3}}{10}$ meters, or 0.160mm.

Again, force density profiles play a role in the size of the vertical grid spacing. Boeuf and Pitchford data require a very limited area for their force density profile, and as such the grid to be used for simulating their profile is built so that 51 points are contained in the first 0.1mm, a spacing of 2 μm. The Roy and Gaitonde profile is again, much larger, and extends to 1cm off of the surface of the flat plate. A total of 101 points are contained in this region for a spacing of 100 μm. The cell spacing for either of these profiles is less than the smallest cell size as determined by the fastest free stream velocity.

Each cell height above the profile areas follows a tanh(x) spacing approach in Gridgen®. The smallest cells are at the bottom of the grid where the DBD is simulated and the largest cells are at the top of the grid where the free stream velocity is.

The flat plate grids were constructed with the specifications for vertical and horizontal grid point spacing as seen in Figure 33 and Figure 34.

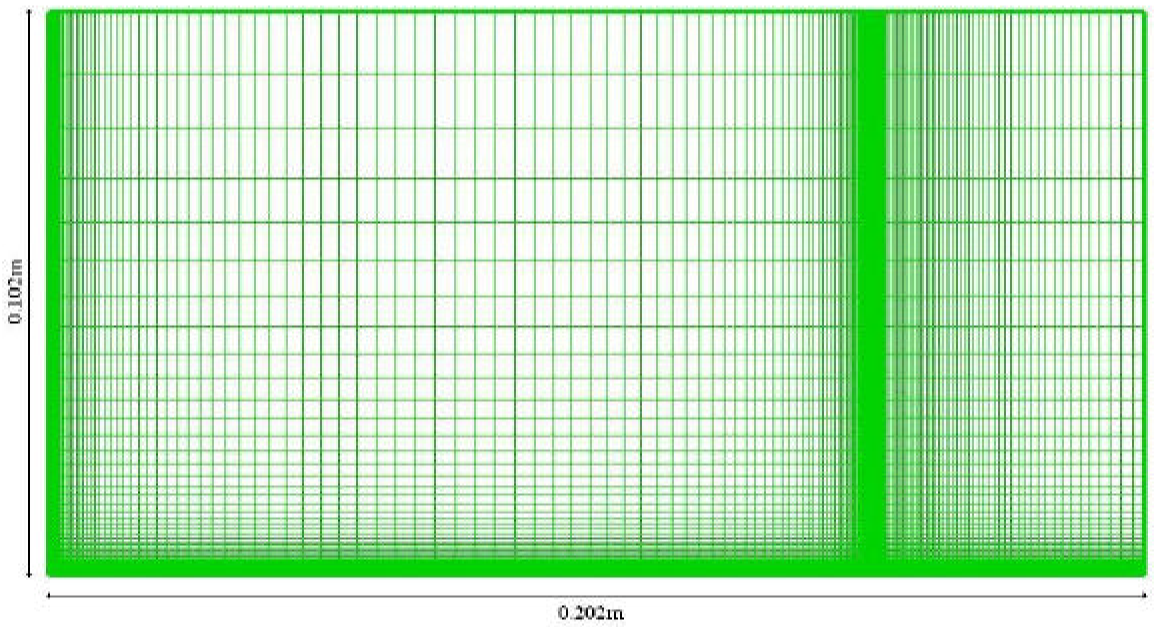


Figure 33
Boeuf and Pitchford Flat Plate Grid Geometry

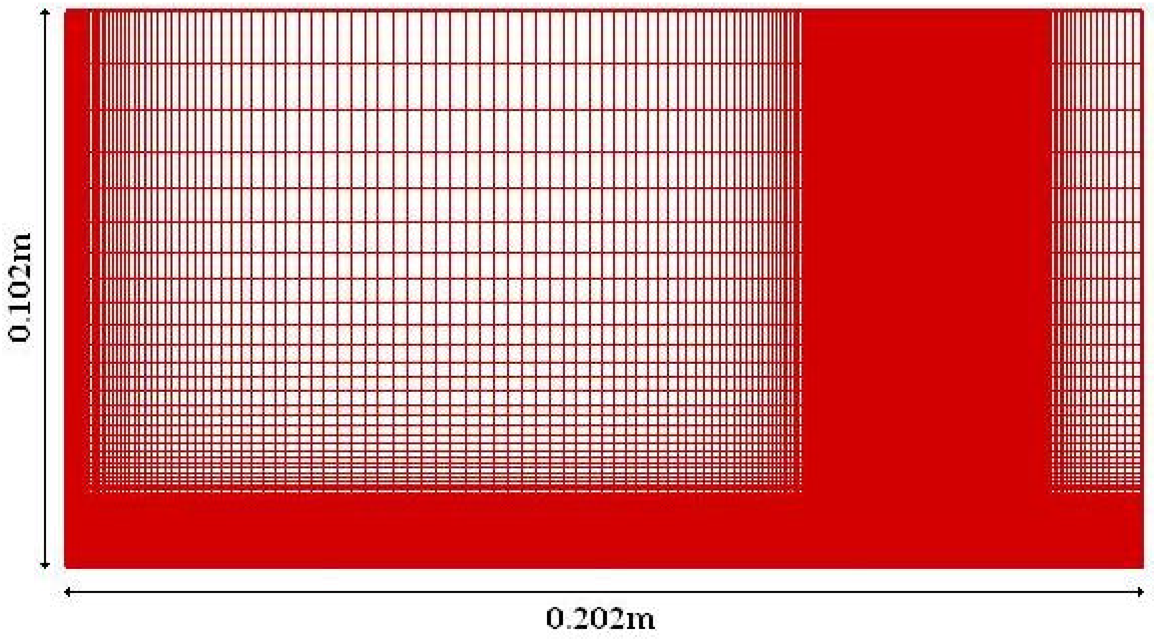


Figure 34
Roy and Gaitonde flat Plate Grid Geometry

Flat Plate Validation

The Flat Plate grids for the Boeuf and Pitchford profile and the Roy and Gaitonde profile were validated by examining the boundary layer profile against a Blasius profile. To obtain the boundary layer velocity profile for each of the grids, an unsteady viscous simulation was run using 0.001 second time steps for 1000 steps (1 second total time) at 2 m/s, 15.2 m/s, and 30.4 m/s. For each time step, a maximum of 20 sub-iterations could be performed before moving to the next time step. The residual was also monitored for convergence to 10^{-6} for each time step, a condition that would cause a move to the next time step prior to 20 sub-iterations being completed. Further details concerning the simulation setup can be found in Appendix C.

NACA 0009 Airfoil Grid Geometry

The creation of the NACA 0009 Airfoil grid involved three data sets, each increasing in fidelity. The grids were constructed in Gridgen®. Airfoil Data Set #1 and #2 found in Table 11 and Table 12, respectively, in Appendix A, were first used in an attempt at validation. However, neither was found to have the fidelity needed for the NACA0009 airfoil shape at the Leading Edge (LE). The next method explored is a mathematical formula [13] which yields much higher fidelity in this region.

Given a 4-digit symmetric airfoil such as the NACA 0009, we can decipher its naming convention to give:

NACA	0	0	0	9		f	maximum camber
NACA	f	x_f	t	t		x_f	position of maximum camber
						x	position along x-axis
						t	$\frac{\text{thickness}}{\text{chord length}}$, the digits represent a %, therefore 09=9%=.09

To locate a 2-D coordinate, the following equations are employed.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \mp y_t \cdot \sin q \\ y_c \pm y_t \cdot \cos q \end{pmatrix} \tag{30}$$

where

$$\tan q = \frac{dy_c}{dx} \tag{31}$$

$$\frac{y_c}{c} = \frac{f}{c}\frac{1}{(1-x_1)^2}\left[(1-2x_1) + 2x_1\frac{x}{c} - \left(\frac{x}{c}\right)^2\right] \quad x_1 = \frac{x_f}{c} \tag{32}$$

$$\frac{y_t}{c} = 5t\left[0.29690x^{0.5} - 0.12600x - 0.35160x^2 + 0.28430x^3 - 0.10150x^4\right] \tag{33}$$

With no camber, f=0 and x_f=0, the above equations simplify to:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ \pm \mathrm{y}_t \end{pmatrix} \tag{34}$$

where

$$\mathrm{y}_t = \mathrm{c} \cdot 5t\left[0.29690x^{0.5} - 0.12600x - 0.35160x^2 + 0.28430x^3 - 0.10150x^4\right] \tag{35}$$

These equations will be used in the UDF to find points along the airfoil, as well as give the tangent slope at a given x-point by deriving the following equation from the above.

$$\mathrm{y}_t' = \mathrm{c} \cdot 5t\left[0.5*0.29690x^{-0.5} - 0.12600 - 2*0.35160x + 3*0.28430x^2 - 4*0.10150x^3\right] \tag{36}$$

The resulting airfoil shape is much smoother than the previous data sets given. The data points are listed in Appendix A. At the leading edge (LE), the difference is the most dramatic. At the trailing edge (TE), the difference is negligible. The overall shape of the airfoil has not changed by employing an equation for a model. However, more fidelity was gained in specific regions, such as the leading edge. A visual comparison between the data sets at the leading and trailing edges was performed in Figure 35 and Figure 36. The increase in data points has given rise to a much smoother curve in both of these areas.

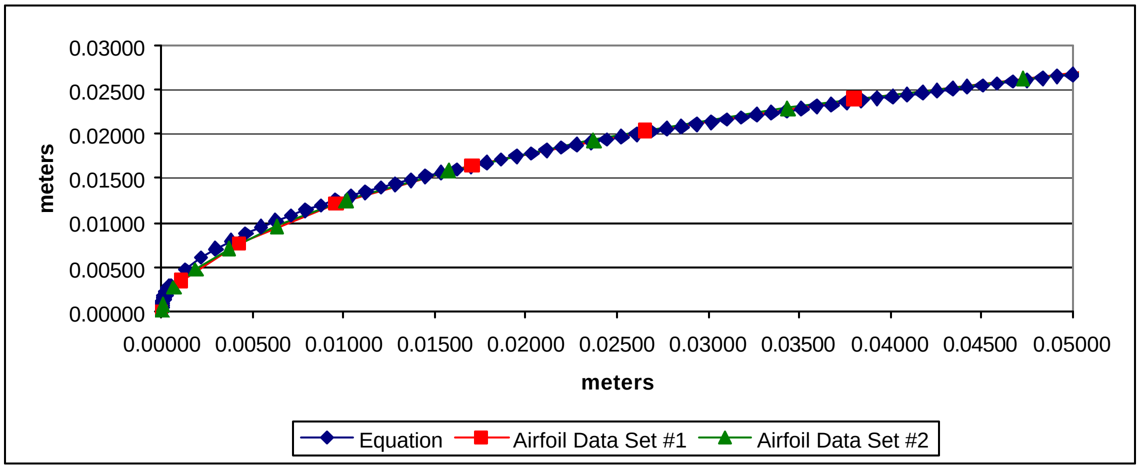


Figure 35
Leading Edge of Airfoil Comparison between Equation and Data Sets

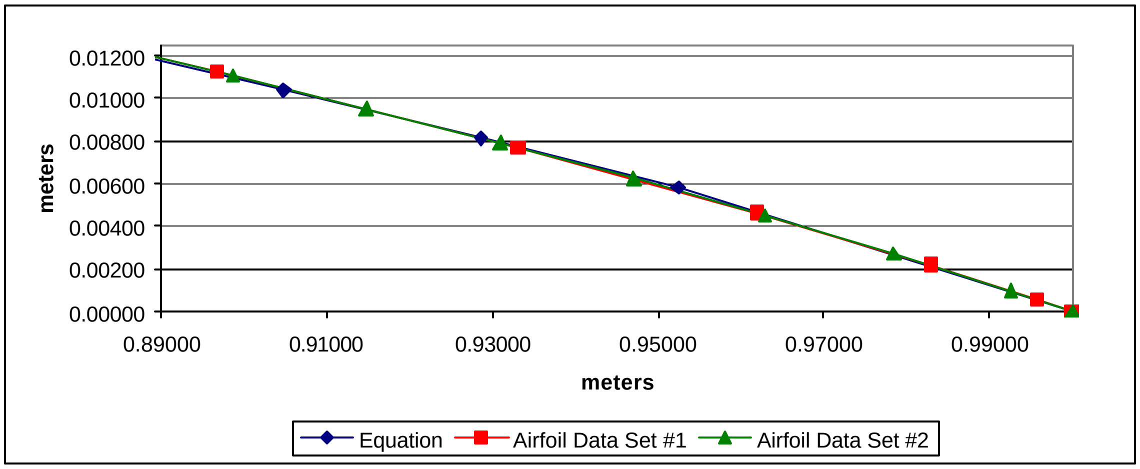


Figure 36
Trailing Edge of Airfoil Comparison between Equation and Data Sets

The airfoil grid was constructed using the parameters of grid spacing for the Roy and Gaitonde flat plate profile. The top section was first constructed and then mirrored to create a symmetric bottom section. The top section was then increased in its number of points and its horizontal grid spacing was altered to fit the horizontal grid spacing of the flat plate. The bottom section of the airfoil remained sparser in points as there was no need for a greater fidelity along the cord of the airfoil. Points were clustered via a “tanh” function along the airfoil curve towards the leading edge for both sections of the airfoil, where points steadily increase in their separation distance for smooth cell size changes. Finally, an elliptical solver was run in Gridgen® to make the cells closest to the boundaries orthogonal. The “tanh” spacing and the elliptical solver have the effect of aiding in solution stability.

The final grid geometry is displayed in Figure 37, showing the increased density of points near the boundary layer and on the top of the airfoil at 75% cord length where the DBD is to be simulated.

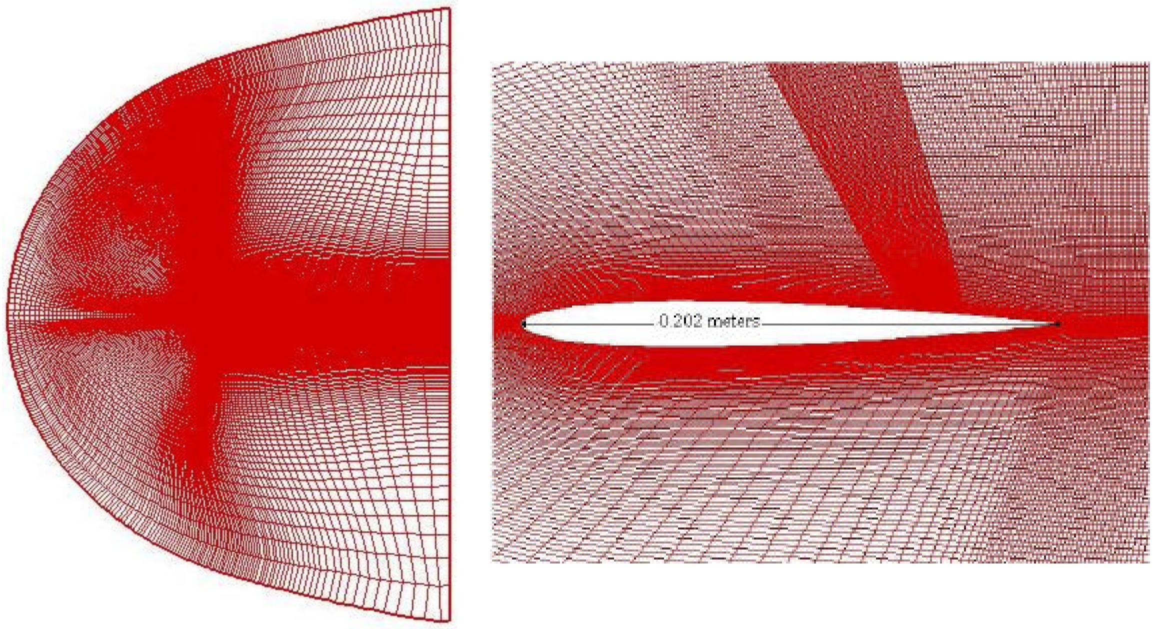


Figure 37
Final NACA 0009 Airfoil Grid Geometry

NACA 0009 Airfoil Validation Confirmation

Experimental data was obtained from Selig [15] and is compared to simulated data obtained for validation of the grid mesh. Data labeled SATurb or Laminar preceding the Reynold's number is Fluent® simulated data. A Spalart-Allmaras Turbulence model was employed to obtain the data shown in Figure 38 thru Figure 41 and in additional data presented in Appendix A in Figure 69 thru Figure 72. As seen in Figure 38 thru Figure 41, the Laminar model data curves did not match as well as the turbulence model data when compared to experiments.

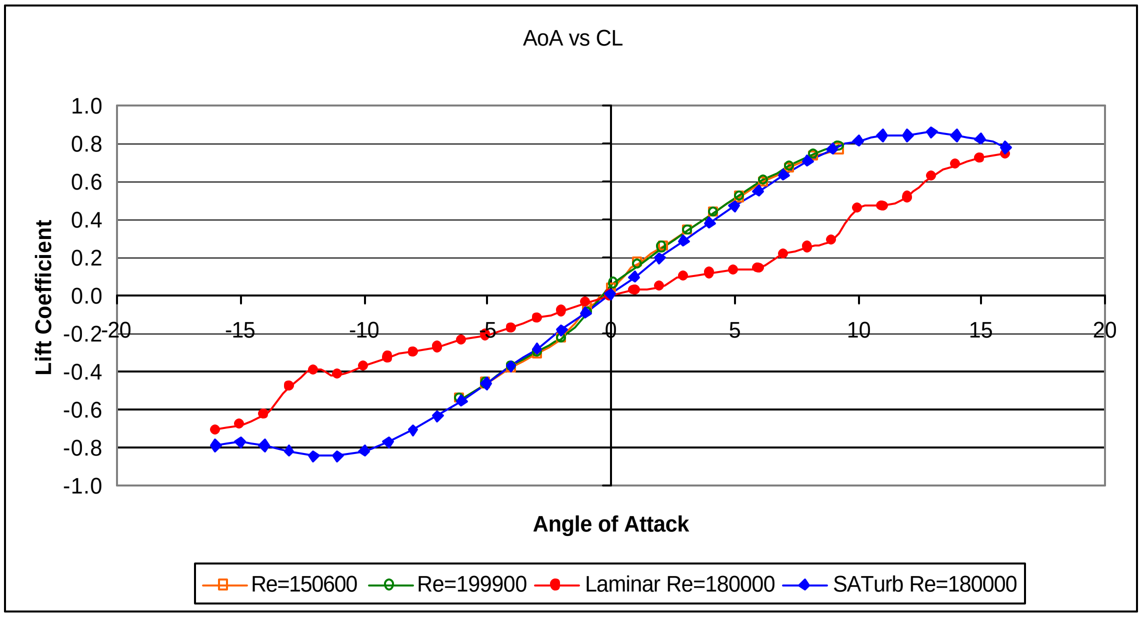


Figure 38
Re_c=180,000
NACA 0009 Angle of Attack VS Lift Coefficient
Comparison of Laminar Model and Spalart-Allmaras Turbulence Model

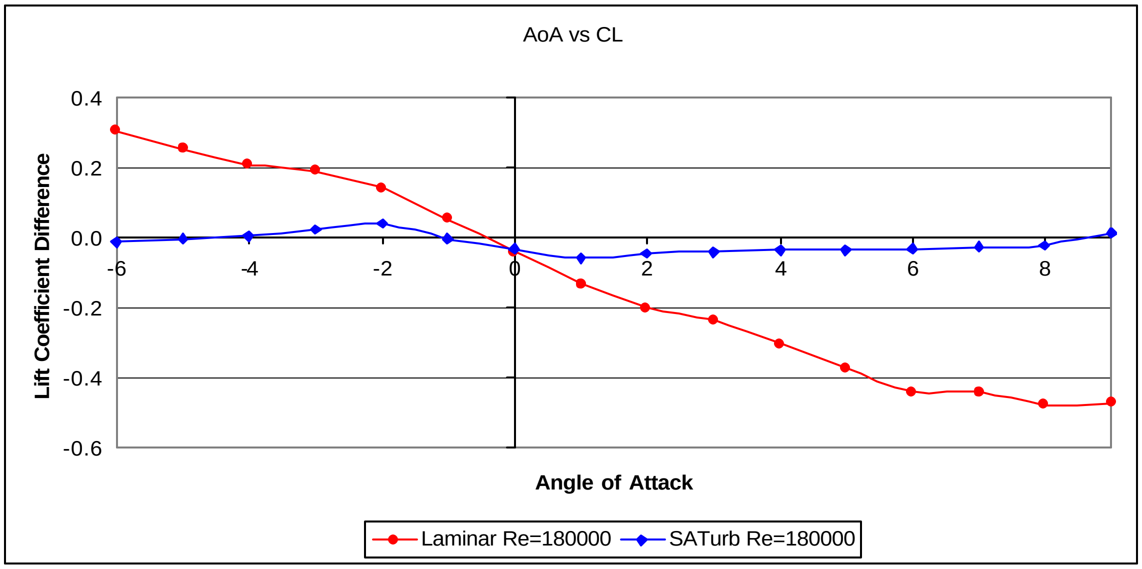


Figure 39
Re_c=180,000
NACA 0009 Angle of Attack VS Lift Coefficient Difference
Comparison of Laminar Model and Spalart-Allmaras Turbulence Model
as Compared to Experimental Data by Selig [15]

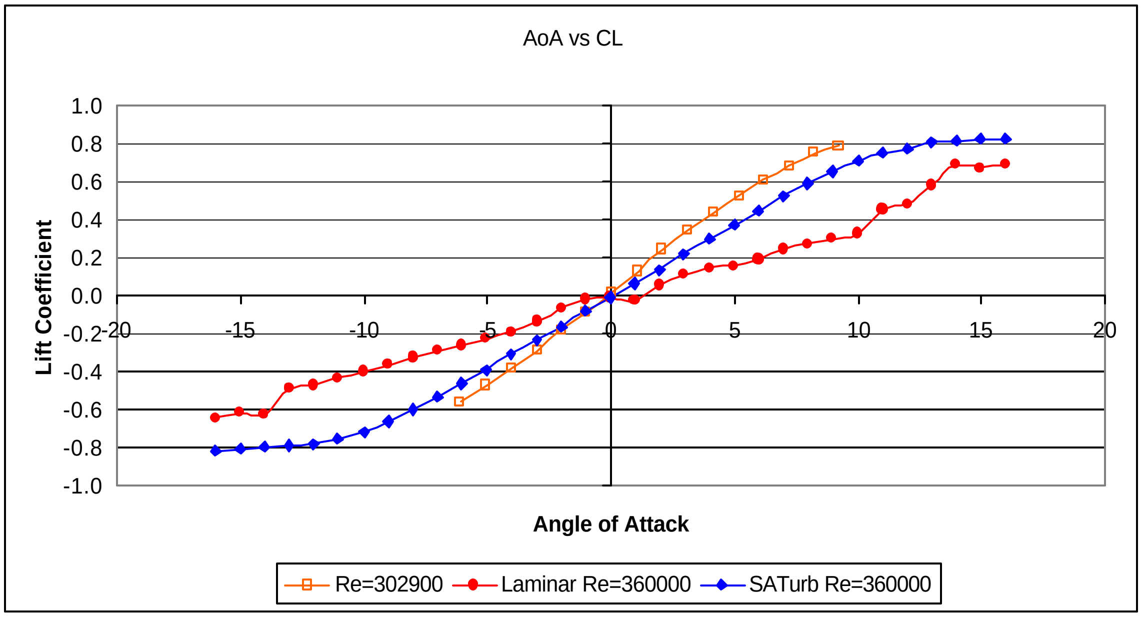


Figure 40

Re_c=360,000

NACA 0009 Angle of Attack VS Lift Coefficient

Comparison of Laminar Model and Spalart-Allmaras Turbulence Model

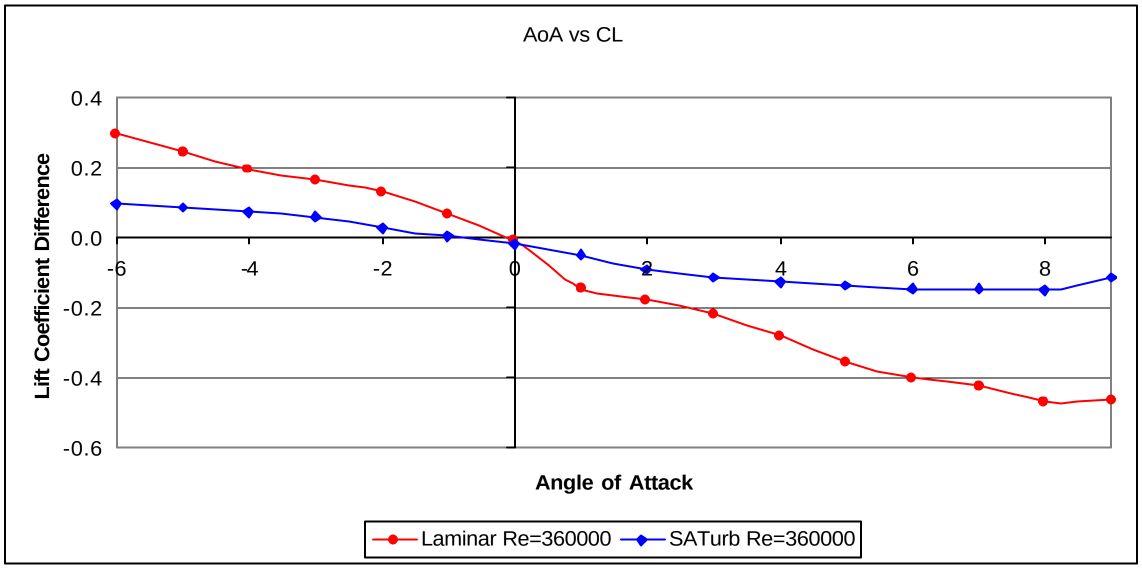


Figure 41

Re_c=360,000

NACA 0009 Angle of Attack VS Lift Coefficient Difference

Comparison of Laminar Model and Spalart-Allmaras Turbulence Model

as Compared to Experimental Data by Selig [15]

The physical data that is displayed in the previous figures (Figure 38 thru Figure 41) was not tripped to turbulent over the airfoil for the purpose of the measurements [15]. This was confirmed with Selig [15] via e-mail correspondence. As a result, it is expected that a Laminar model should be applied. Further, turbulence is usually calculated to occur above a Reynolds' number of 0.5×10^{6}. However, there is indication that turbulence and separation are occurring at the half-cord on this symmetric airfoil. The Reynolds' numbers may be approaching a lower limit for consideration of a turbulence model such as Spalart-Allmaras. Figure 39 and Figure 41 show that the Spalart-Allmaras data is much more consistent with Selig's experimental data as compared with the laminar data.

The turbulent simulated data obtained more closely mirrors the physical data given by independent sources [15]. This suggests that the model is best represented using the Spalart-Allmaras Turbulence Model in Fluent®. However, because turbulence was not tripped during the actual physical measurements, this validation can not be substantiated and a laminar solver is expected to be run. Nonetheless, the laminar solver will not be run due to the poor behavior of the model as compared to measured experimental data. Instead, for purposes of this simulation, it is assumed that the Spalart-Allmaras turbulence model will apply so as to give data that more closely mirrors reality.

IV. Results

It was necessary to examine different pieces of the DBD simulation's behavior in order to arrive at a parameterized view of the DBD's operation. Several data sets were simulated and results returned. Thermal energy addition versus momentum addition effects were simulated as well as power per unit length effects. This section will cover these simulation sets and their results.

Processing

The airfoil data sets simulated required a great deal of processing power. Each set of data for the airfoil was simulated at angles of attack from 0° to $+16^\circ$ in 2° increments for each of the 2 Reynolds' numbers used in validation; a total of 18 simulations for each set. The simulations were processed on the AFIT computer cluster machine Tahoe.

Tahoe is a 64 node computer cluster with each node having 2 AMD Opteron 2.2GHz processors. The node set of processors reside on the same control board and have 4GB of RAM shared in a NUMA, Non-Uniform Memory Architecture, where each processor on that node is guaranteed its part of the RAM, not shared. Connections between the nodes are handled using 1Gbps Ethernet.

AFIT has ownership of 30 Main/Startup Fluent® licenses with 88 Multi-processor Fluent® licenses. The main Fluent® licenses along with available nodes drove the overall rate at which each set was completed. Each simulation was performed on 1 node, using both processors in an attempt to reduce computational time. The average completion time for a simulation was 12 hours, with each set taking 24 hours to complete. The validation cases and another set were run using only single processors, which were found to be 1.5 times faster. This was discovered after most of the simulations had been run and is not a surprising result for such a small 2D simulation. The decreased performance with increased processor numbers is due to the network transfer speed being a significant factor in the processing of each iteration; which means the processors are idle while they transfer information between each other and are therefore less efficient. Future work should use only a single processor for each simulation as a result.

Appendix C details the PBS (Portable Batch System) script and journal files that allowed automation of each simulation.

Matching Force Density Profiles

Several trials were conducted in order to fit the force density profiles similar to those reported by either Boeuf and Pitchford in Figure 18 and Figure 19 or Roy and Gaitonde in Figure 23 and Figure 24. This was due to the boundary layer velocities affecting the profiles after steady state had been reached. As was discussed in the Simulation Setup section, an increase in the force density increases the velocity of the flow, but the velocity increase is indirectly proportional to the flow velocity where the force density is acting. The boundary layer has an increasing velocity as the distance from the wall is increased; corresponding to a force density that drops as the distance from the wall increases.

Velocity profile data reported from these simulations gives its location relative to the overlap between the electrode transition, also reported as the center of the DBD.

Boeuf and Pitchford Force Density Profile

The Boeuf and Pitchford profile has no appreciable effects on the flow at low velocities of 2 m/s, and as a result, less of an effect at higher velocity flows. Figure 43 to Figure 46 show that within the boundary layer there is minimal deviation from the baseline boundary layer velocity profile. A wall jet resulting in deviation from the baseline case is expected as far away as 75mm from the DBD source, as is seen in Figure 3. The electrode length in Figure 3 is 2 orders of magnitude greater than that of the Boeuf and Pitchford simulation. The simulated DBD force density, shown in Figure 42, must be greater in this spatially confined profile in order to be effective in its confined volume.

Table 3. Boeuf and Pitchford Matching Data Set

$\frac{\text{Power (W)}}{\text{1 meter}}$	Thermal Energy %	Thermal Power $\frac{W}{m}$	Momentum Power $\frac{W}{m}$	X-Momentum		Y-Momentum	
				%	Power $\frac{W}{m}$	%	Power $\frac{W}{m}$
0.0003	10%	0.00003	0.00027	100%	0.00027	0%	0

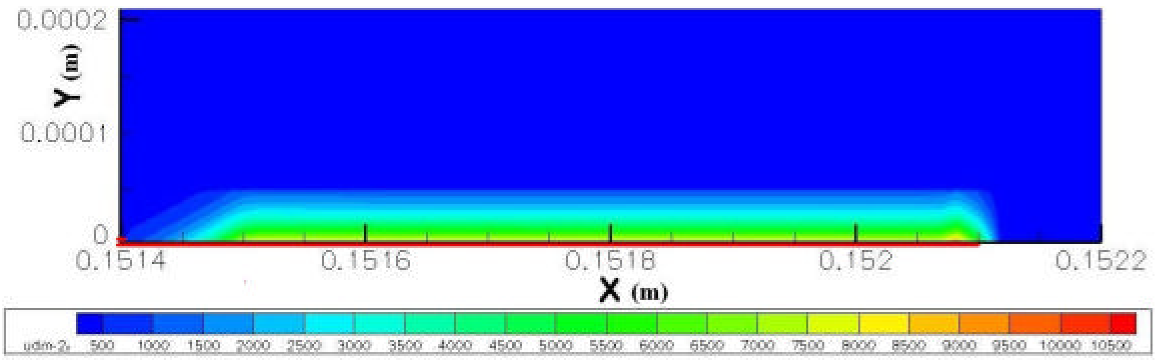


Figure 42
X-Momentum Force Density for the Case in Table 3

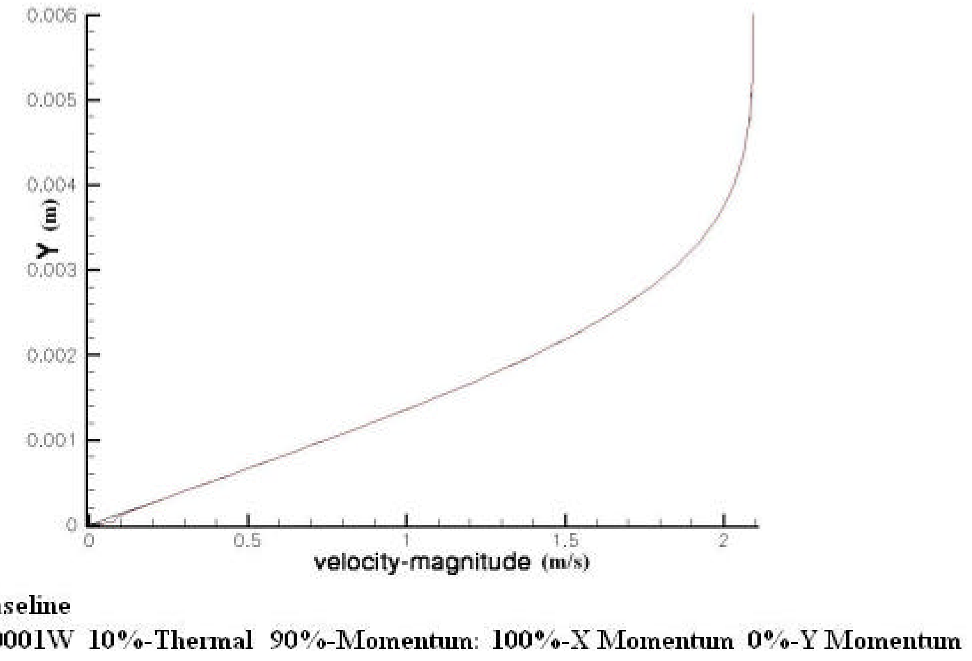


Figure 43
Boeuf and Pitchford Weighting Profile 2 m/s Simulation Result at x=0mm
Compared to Baseline

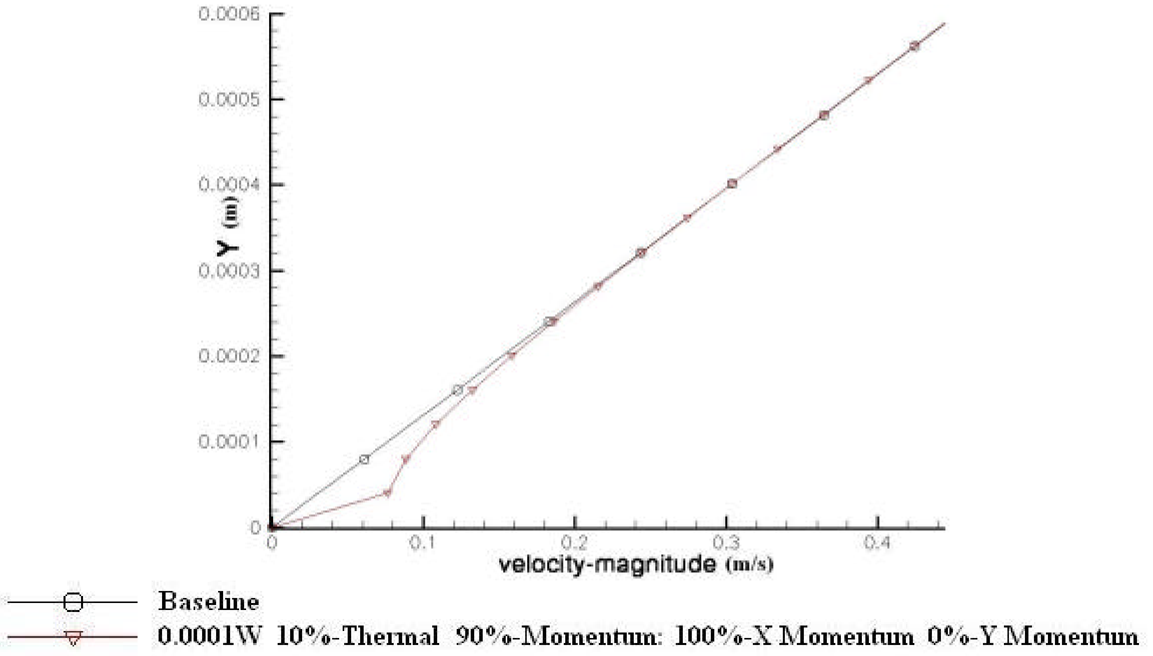


Figure 44
Boeuf and Pitchford Weighting Profile 2 m/s Simulation Result at x=0mm
Compared to Baseline (Close-up)

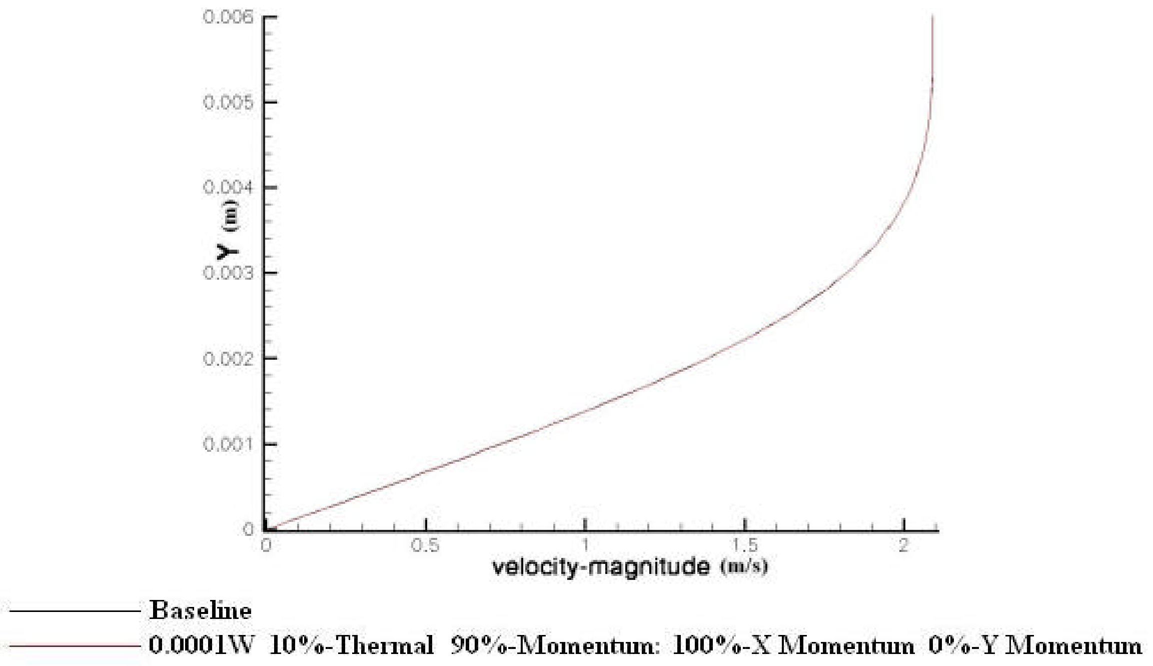


Figure 45
Boeuf and Pitchford Weighting Profile 2 m/s Simulation Result at x=5mm
Compared to Baseline

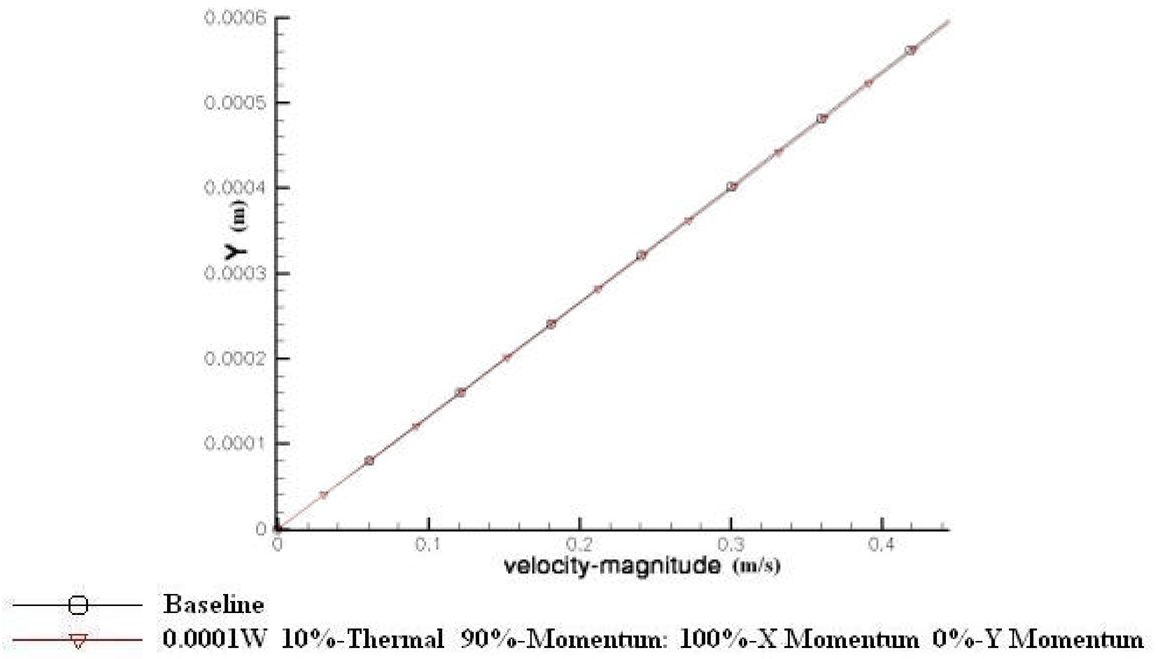


Figure 46
Boeuf and Pitchford Weighting Profile 2 m/s Simulation Result at x=5mm
Compared to Baseline (Close-up)

Several cases were examined using the force density profile of an extremely confined volume by which to represent the DBD. Figure 48 displays these simulation results and compares against the baseline case at x=5mm from the DBD location. The simulations' specific settings are listed in Table 4. These settings resulted in the force density shown in Figure 47 and represent a 2 order of magnitude increase from the values suggested by Boeuf and Pitchford.

Table 4. Boeuf and Pitchford Data Set for Increasing Thermal Addition with Fixed Momentum Percent Addition

$\frac{\text{Power (W)}}{\text{1 meter}}$	Thermal Energy %	Thermal Power $\frac{W}{m}$	Momentum Power $\frac{W}{m}$	X-Momentum		Y-Momentum	
				%	Power $\frac{W}{m}$	%	Power $\frac{W}{m}$
5.0	40%	2.0	3.0	70%	2.10	30%	0.90
	50%	2.5	2.5		1.75		0.75
	60%	3.0	2.0		1.40		0.60
	70%	3.5	1.5		1.05		0.45

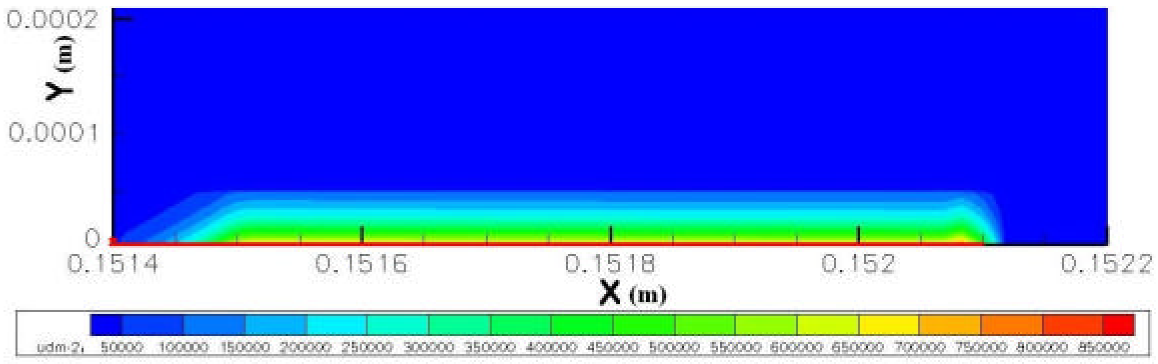


Figure 47
X-Momentum Force Density for 40% Thermal Energy % in Table 4

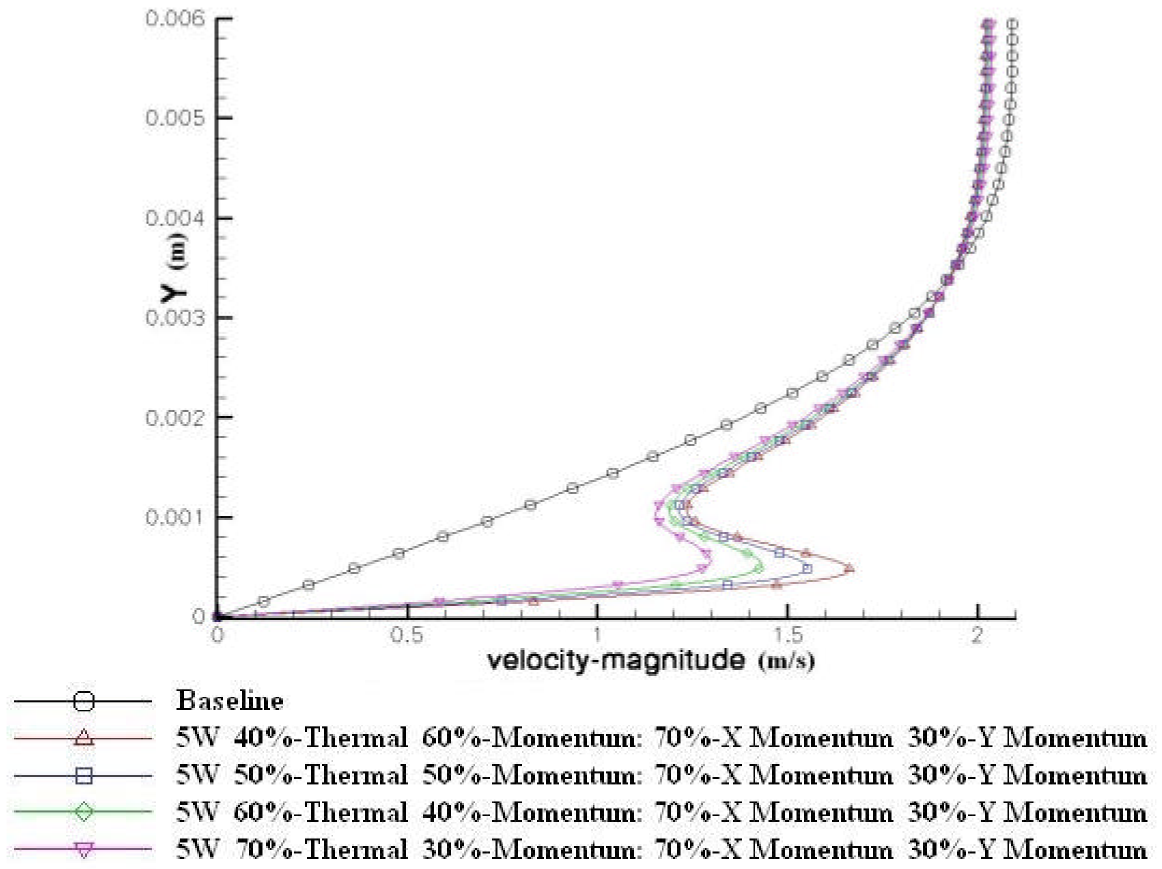


Figure 48
Boeuf and Pitchford Force Density Simulation Result at x=5mm
Compared to Baseline for Test Cases in Table 4

As the amount of thermal energy was increased and therefore the momentum energy decreased, the boundary layer profiles in Figure 48 decrease towards the baseline case as expected. The maximum velocity magnitude is found at the DBD source for these simulations reaching close to 6 m/s, but rapidly falls back to the baseline velocity profile. What was expected from Boeuf and Pitchford's original force density profile was a flow with a wall jet that has a maximum velocity of approximately 8 m/s to 10 m/s as was shown in Figure 20 or a minimum of 2.3 m/s as shown in Figure 28. Boeuf and Pitchford [16] state that their time integrated force density profile will "…approximately give the contours of constant velocity increment in units of m/s". There is a discrepancy between the ~8 m/s wall jet predicted by Boeuf and Pitchford, the 1.9 m/s predicted with stationary flow energy addition analysis in the Simulation Setup, and the 0.15 m/s that occurred in the Fluent® simulation. The discrepancy cannot be explained at this time.

In a final attempt to explain this velocity discrepancy, a final set of simulations were performed to examine the added effect of increasing the extent of the force density profile in the horizontal direction. The four cases examined are listed in Table 5. It was hypothesized by Boeuf and Pitchford [16] that their simulation space was not large enough and that by increasing its horizontal extent, the force density would continue. This is a conclusion that can be drawn from examining both impulse density results for their 400μm and 800μm wide DBD simulations; they behave similarly but with a larger horizontal extent for the force density when examining the 800μm case. Therefore, if the horizontal extent of the DBD's force density was allowed to increase, the flow would undergo more accumulated impulse and have a larger velocity increase. Initially, in Figure 50, the velocities begin with near the same wall jet velocity increase at x=0mm. In the following figures, Figure 51, Figure 52, and Figure 53, velocity profiles are taken an additional 5mm from x=0mm for each subsequent figure. It is apparent that there is a small wall jet velocity increase due to the lengthened horizontal spatial extent. However, as soon as the velocity profile is no longer being measured within a force density profile, the wall jet abruptly returns to baseline velocity profile behavior. This is still not the suggested behavior. It may be possible that the vertical extent of the force density will also need to be increased, such that a tendency is more towards a force density profile of Roy and Gaitonde's reports.

Table 5. Boeuf and Pitchford Data Set for Increasing Simulation Extent while maintaining Force Density Profiles Suggested by Boeuf and Pitchford

$\frac{\text{Power (W)}}{\text{1 meter}}$	Simulation Extent (mm)
0.0003	0.8
0.0005	1
0.0025	5
0.0050	10

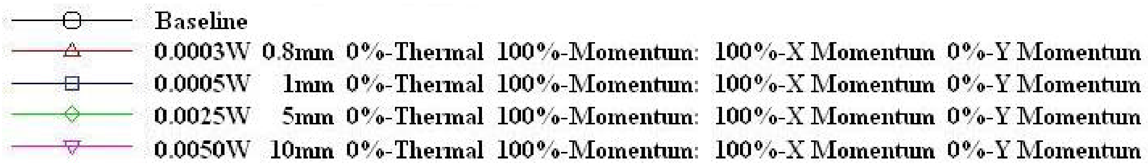


Figure 49
Legend for the Velocity Profiles Listed in Table 5 and Displayed in Figure 50 to Figure 53 (Close-up)

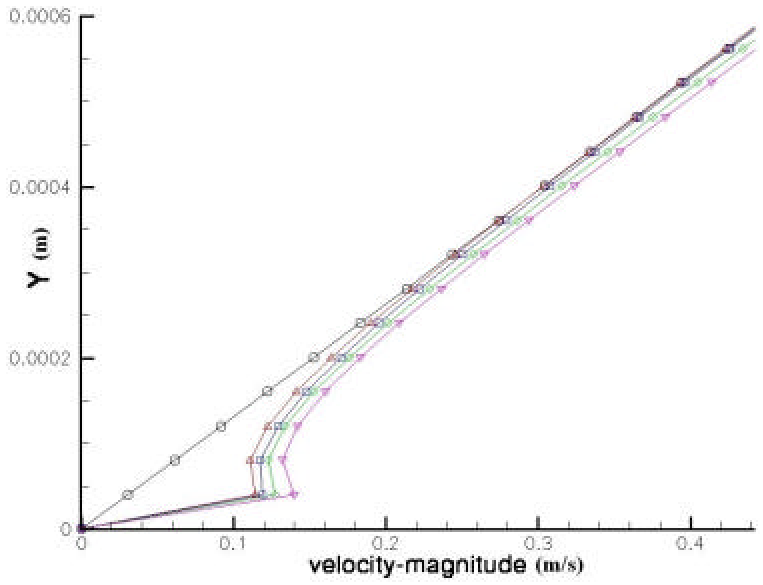


Figure 50
Boeuf and Pitchford 2m/s Increasing Simulation Extent Velocity Profiles at x=0mm Compared to Baseline for Test Cases in Table 5 (Close-up)

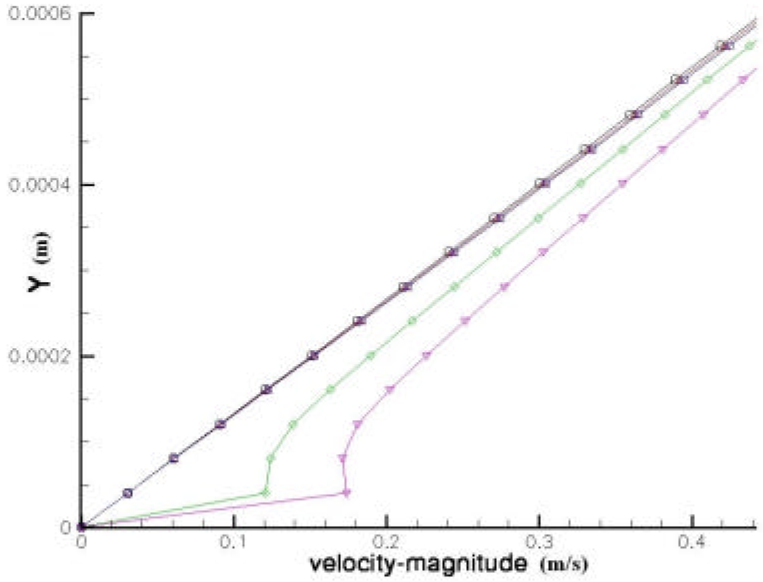


Figure 51
Boeuf and Pitchford 2m/s Increasing Simulation Extent Velocity Profiles at x=5mm Compared to Baseline for Test Cases in Table 5 (Close-up)

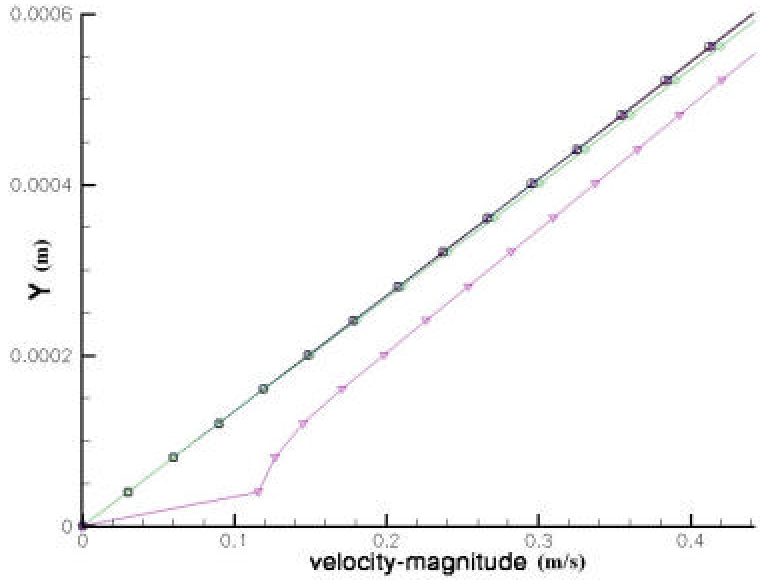


Figure 52
Boeuf and Pitchford 2m/s Increasing Simulation Extent Velocity Profiles at x=10mm Compared to Baseline for Test Cases in Table 5 (Close-up)

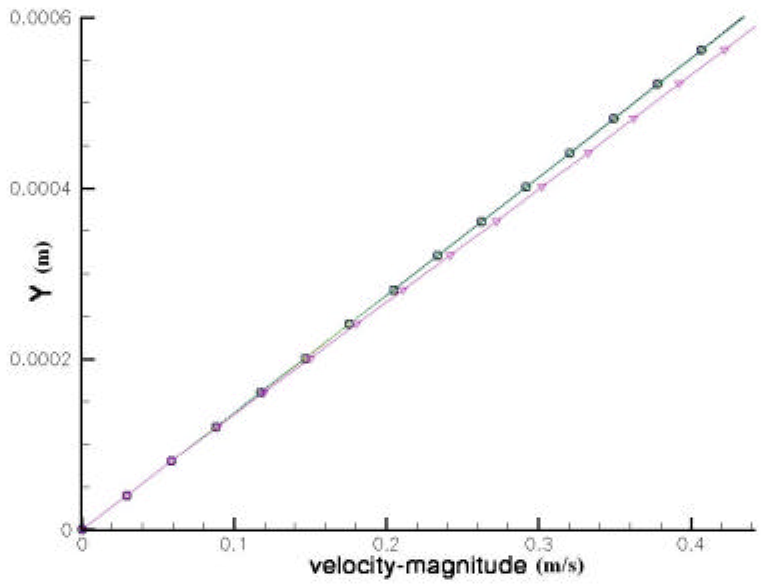


Figure 53
Boeuf and Pitchford 2m/s Increasing Simulation Extent Velocity Profiles at x=15mm Compared to Baseline for Test Cases in Table 5 (Close-up)

Roy and Gaitonde Force Density Profile

The Roy and Gaitonde profile has effects at a free stream velocity of 2 m/s. This section details the required settings and results of the Roy and Gaitonde force density profile on a flat plate with a free stream velocity of 2 m/s.

The setup that allows for the given profile is listed in Table 6. The two figures that follow the table, Figure 54 and Figure 55, show that the force density profile was simulated correctly with the settings in Table 6. This is not entirely evident in Figure 54 due to the negative region at 0.16m having less of a peak absolute magnitude, and therefore not showing in the same fidelity as the positive regions.

The wall jet is evident at x=5mm as seen in Figure 56 and continues downstream in an accelerating manner to a peak of 4.2 m/s at 10mm from the DBD source. The wall jet's velocity magnitude then starts to fall off due to fluid sheer, but remains near 4 m/s at 50mm from the DBD source. This behavior is nearly twice the expected velocity estimated in the Simulation Setup section as well as depicted in Figure 25 with experimental DBD setup and measurements, but fulfills the original goal of producing a near 5 m/s wall jet.

Table 6. Settings to Achieve Roy and Gaitonde Weighting Profile

$\frac{\text{Power (W)}}{\text{1 meter}}$	Thermal Energy %	Thermal Power $\frac{W}{m}$	Momentum Power $\frac{W}{m}$	X-Momentum		Y-Momentum	
				%	Power $\frac{W}{m}$	%	Power $\frac{W}{m}$
5.0	97.8%	4.89	0.11	90%	0.099	10%	0.011

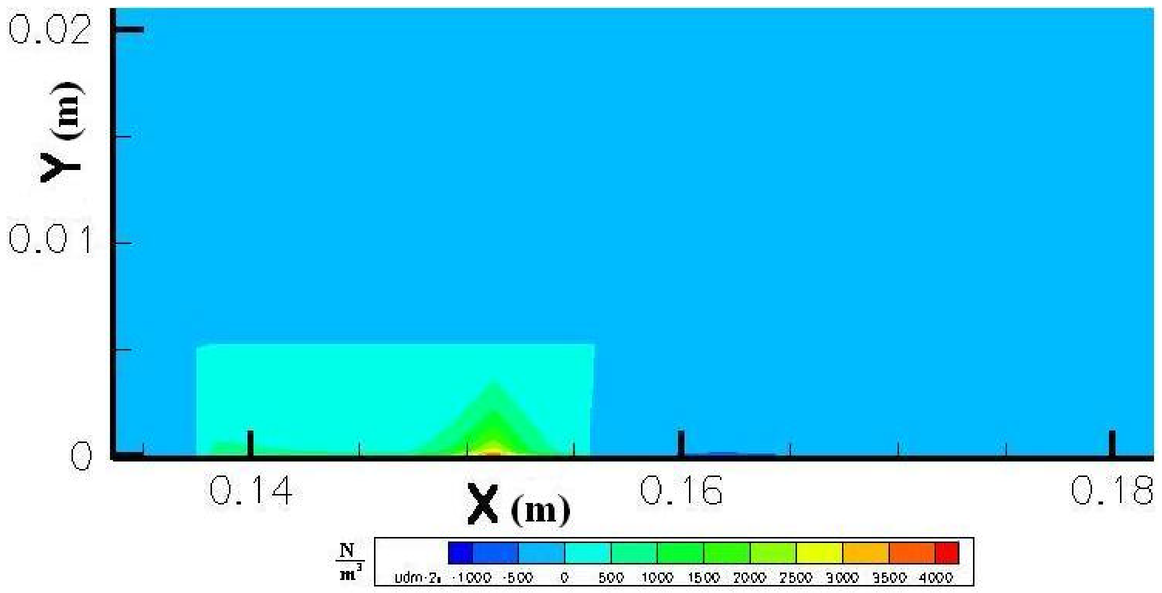


Figure 54
Roy and Gaitonde X-Momentum Force Density Profile from Table 6

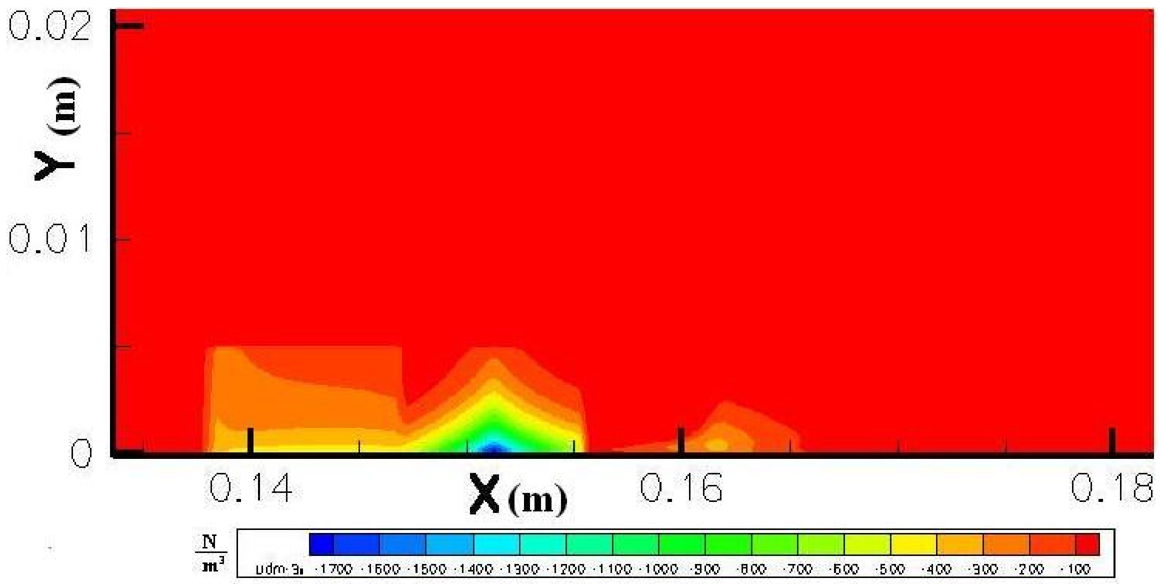


Figure 55
Roy and Gaitonde Y-Momentum Force Density Profile from Table 6

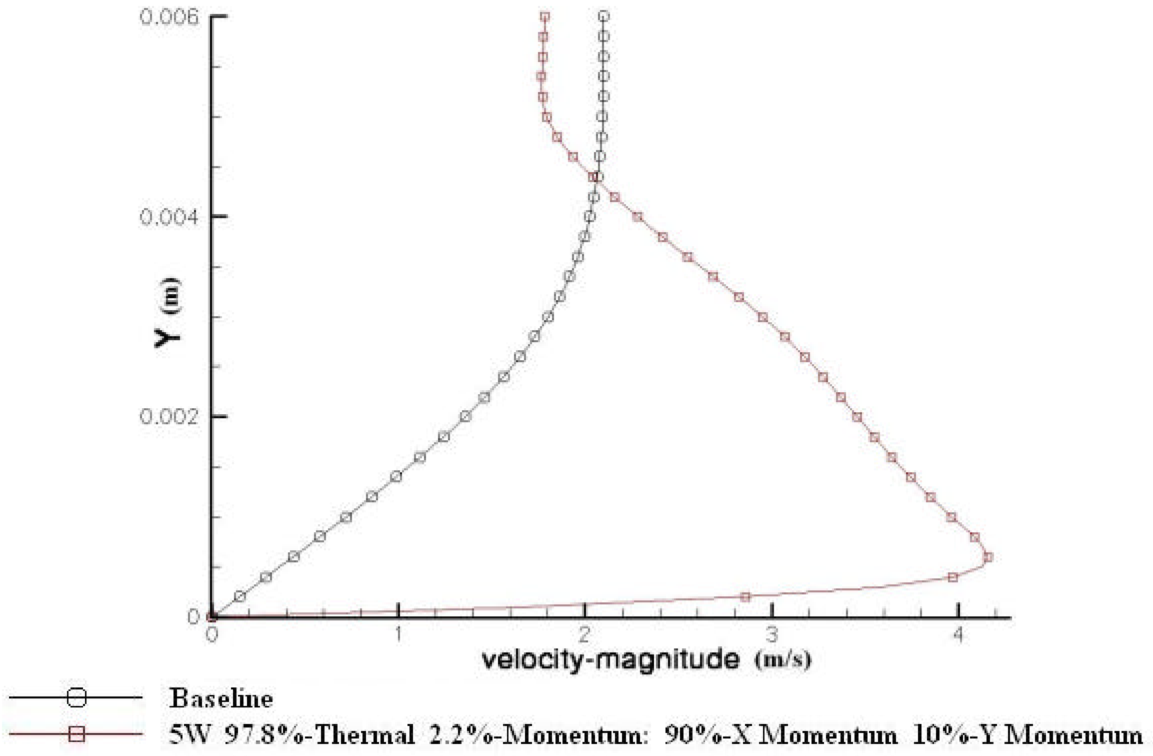


Figure 56
Boundary Layer Velocity Magnitude Profile for
Roy and Gaitonde Force Density Simulation Result at x=5mm
Compared to Baseline for Test Case in Table 6

Thermal Energy Dependence

It has been noted by Newcamp [18], that there is a limit to effectiveness of the DBD input power. Newcamp found that as the power was increased, and the flow velocity remained unchanged, the velocity of the wall jet decreased. The suspect mechanism is that the amount of momentum transfer to the flow remains unchanged while the thermal energy addition increases. For a gaseous fluid, increasing its temperature will result in a fluid that is more viscous, or more difficult to move. This is the opposite effect that temperature increase will have on liquid fluids.

To confirm or deny the effects of thermal energy being the mechanism for slowing down the jet, several simulations were run that fixed the momentum sources in both the x and y directions, yet allowed for the thermal energy addition to be varied in a 2 m/s freestream flow over a flat plate. The specific settings are displayed in Table 7. The Roy and Gaitonde force density profile was used, as it has appreciable effects on the boundary layer flow as compared to the Boeuf and Pitchford force density profile.

The high and the low end of input power for Table 7’s range was derived using the near minimum amount of power it would take to create an approximate 5 m/s flow and the maximum that was put into a DBD before it physically failed and could never be used again from Newcamp’s experiments [18]. Newcamp reports that failure occurred at $25\text{W}/5'' \approx 200 \text{ W/m}$.

The boundary layer ends 6mm above the surface of the flat plate for a 2 m/s flow. The boundary layer profile and the profiles of the simulations run, listed in Table 7, are displayed in Figure 57. As expected, Figure 57 shows that as the thermal energy is increased, the wall jet velocity will decrease. However, it is also seen that this decrease is negligible, varying velocity from 4.6 m/s to 4.4 m/s, a 0.2 m/s difference that decreases as the thermal power addition is increased over 4 orders of magnitude. Further, Figure 59 shows that the linear temperature increase seen is consistent with estimated calculations using the equation $\Delta T = \frac{\Delta e}{\frac{5}{2}k_b N}$; where $k_b = 1.380658 \cdot 10^{-23}$ is the Boltzmann Constant, $\Delta e = \text{Power} \cdot \vartriangle t$ where $\vartriangle t$ is the user defined time step taken by Fluent® to resolve a time accurate solution, and $N = r \cdot \frac{1 \text{ mole}}{0.029kg} \cdot \frac{6.0221367 \cdot 10^{23} \#}{1 \text{ mole}} \cdot \text{vol}$.

This simulation set from Table 7 does account for a decrease in velocity magnitude as the input power is increased; as described by Newcamp [18]. However, to be consistent with Newcamp's reporting in Figure 58, a more accurate scale to compare on would have Figure 57's data divided by U_∞, the fluid's free stream velocity. As $\mathrm{U}_\infty = 2 \text{ m/s}$, the peak magnitudes would then read 2.3 and 2.2 for the lower and higher thermal energy cases, respectively. This is a 0.1 difference that is similar to the data presented in Figure 58.

Table 7. Thermal Energy Dependence Test Case Sets for Roy and Gaitonde Profile

$\frac{\text{Power (W)}}{\text{1 meter}}$	Thermal Energy %	Thermal Power $\frac{W}{m}$	Momentum Power $\frac{W}{m}$	X-Momentum		Y-Momentum	
				%	Power $\frac{W}{m}$	%	Power $\frac{W}{m}$
0.2	10%	0.02	0.18	55%	0.099	45%	0.011
1	82%	0.82					
2	91%	1.82					
3	94%	2.82					
4	95.5%	3.82					
5	96.4%	4.82					
10	98.2%	9.82					
15	98.8%	14.82					
20	99.1%	19.82					
25	99.28%	24.82					
80	99.775%	79.82					
120	99.85%	119.82					
160	99.8875%	159.82					
200	99.91%	199.82					

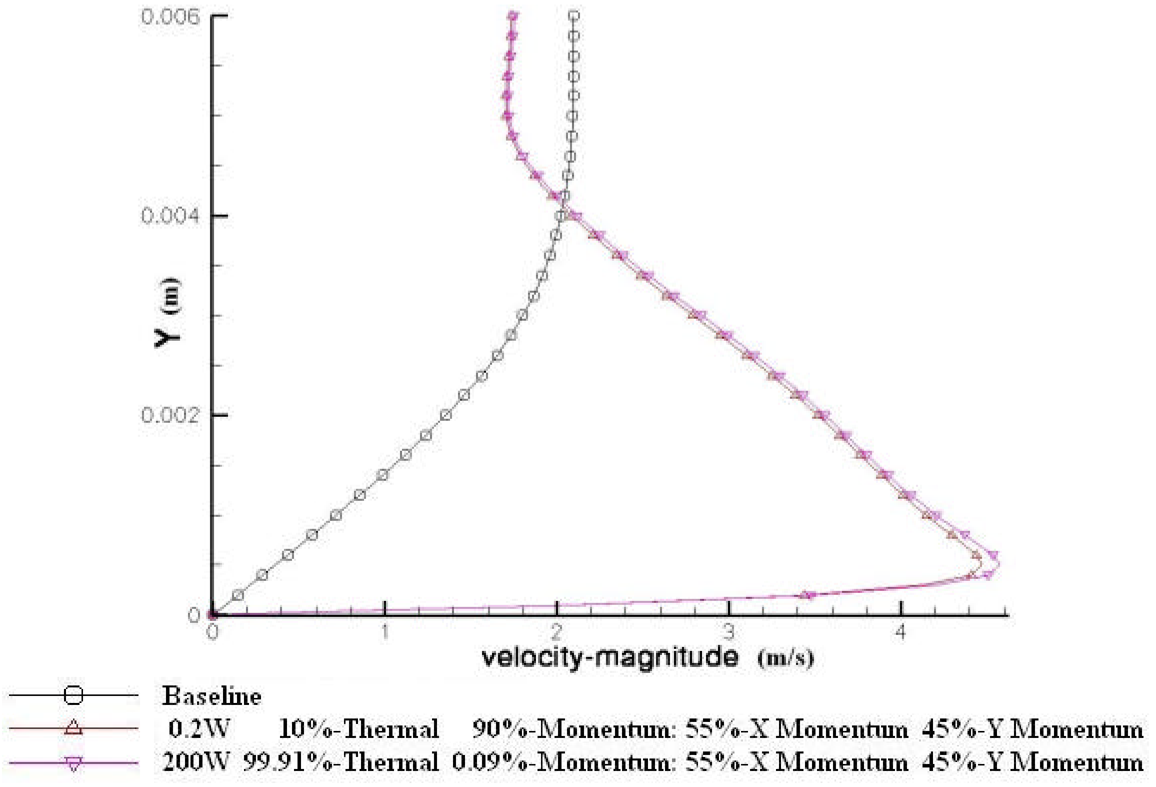


Figure 57
Velocity Profile on Flate Plate 5mm Downstream of DBD Upper Electrode;
Fixed Momentum, Varying Thermal, Re_x=20.8k

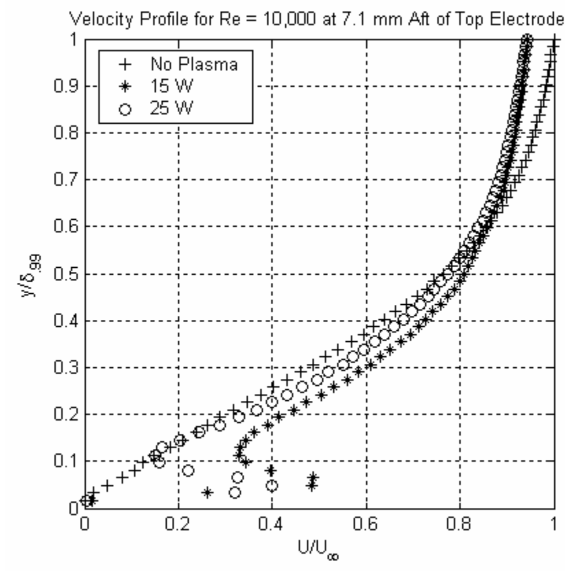


Figure 58
DPIV Velocity Profile 7.1mm Downstream of DBD Upper Electrode [18];
Varying Total Power, Re_x=10k

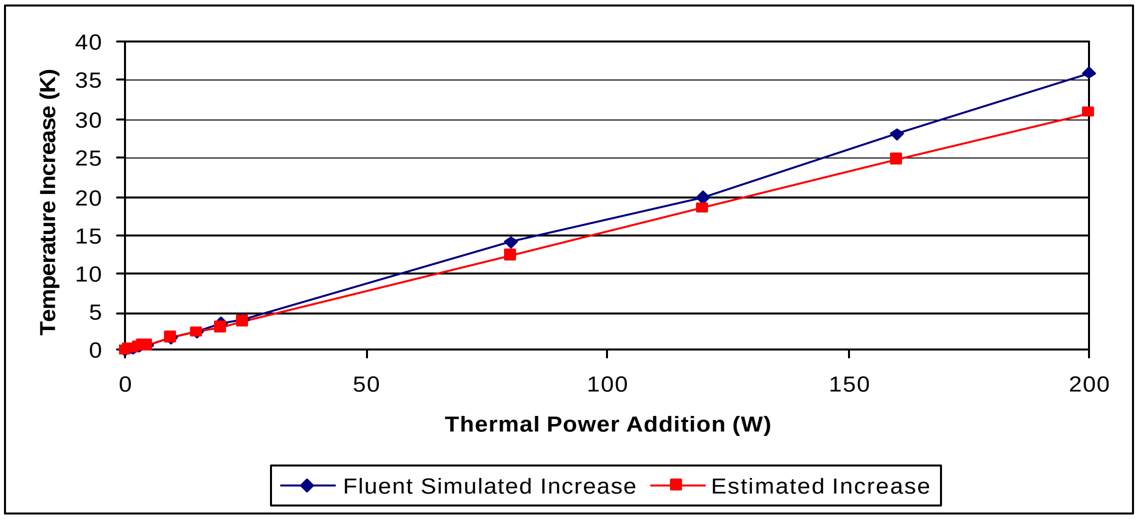


Figure 59
Temperature Increase vs Thermal Power Addition for Table 7 Simulation Set

To be complete, a simulation set with a fixed percentage of power transferred into thermal addition as well as momentum addition with varying power levels was run. The simulation set from Table 8 was performed on the basis that the power level is increased by raising the voltage of the system. The increased voltage does cause more ionization of the fluid, air, and therefore a higher current level of the system. Since $P = I \cdot V$, this leads to the increase in power seen in the experiments. Because the current is increasing, it is not unreasonable to assume that there will be more ionization resulting in a larger wall jet velocity magnitude. This theory would result in the opposite of what is seen in Figure 58. The hypothesis is correct in stating that as power is increased, the wall jet velocity magnitude will increase as is shown in Figure 60. The largest wall jet velocity magnitude coincides with the most power input to the system, with the smallest wall jet velocity magnitude coinciding with the least amount of power input to the system, respectively. However, this is not what is seen in Figure 58, and as such this partition must be discarded. As was shown in the thermal energy dependence simulations just prior to this subsection, there is minimal effect to the wall jet velocity by increasing the thermal energy source input to the system. However, a small variance in the momentum source will lead to a significant change in the wall jet velocity.

Table 8. Roy and Gaitonde Weighting for Varying Power Levels with Constant Percent Momentum and Thermal Addition

$\frac{\text{Power (W)}}{\text{1 meter}}$	Thermal Energy %	Thermal Power $\frac{W}{m}$	Momentum %	Momentum Power $\frac{W}{m}$	X-Momentum %	X-Momentum Power $\frac{W}{m}$	Y-Momentum %	Y-Momentum Power $\frac{W}{m}$
80	97.8%	78.24	2.2%	1.76	90%	1.584	10%	0.176
120		117.36		2.64		2.376		0.264
160		156.48		3.52		3.168		0.352
200		195.60		4.40		3.960		0.440

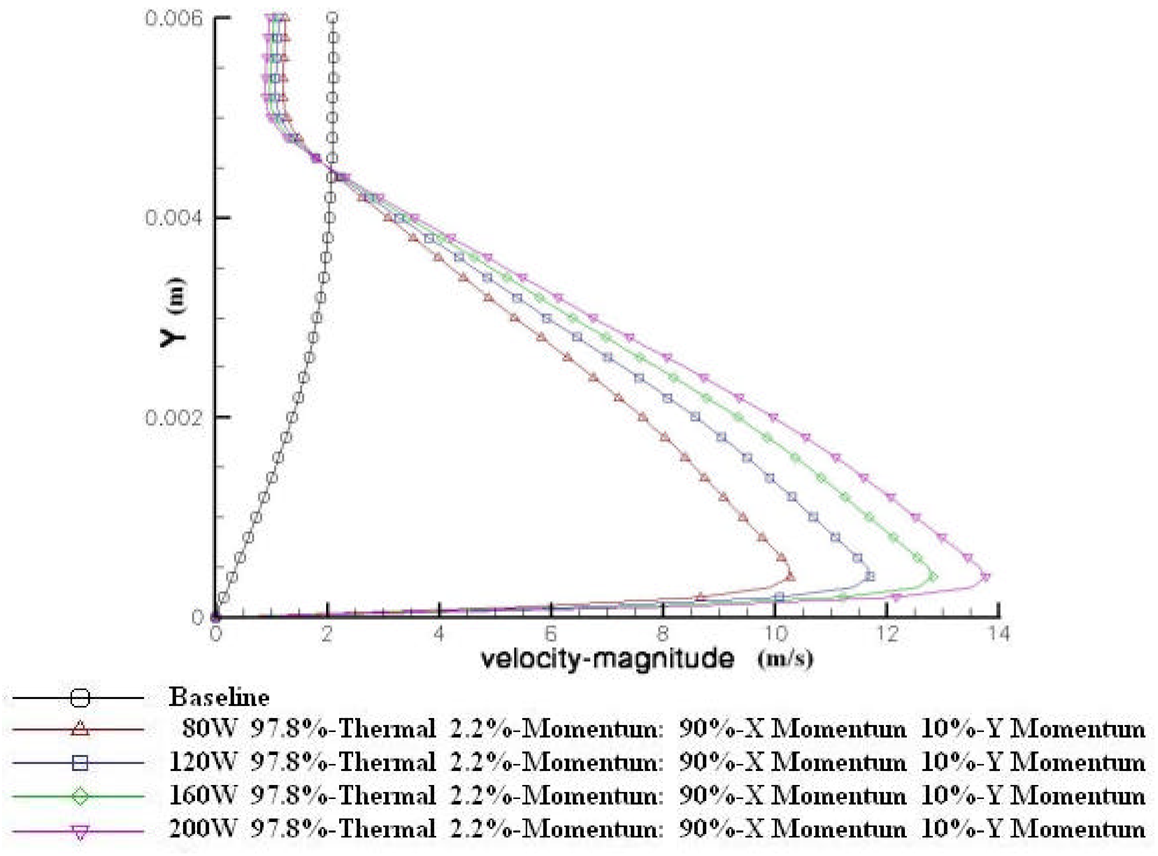


Figure 60
Boundary Layer Velocity Magnitude Profile for
Roy and Gaitonde Weighting at x=5mm for Table 8

Airfoil Results

The airfoil test is the final set of cases. The flat plate scenarios have allowed for the area under examination to be narrowed to a specific weighting profile that best represents the physical effect, as well as how to setup that profile. This set of cases will allow for insight into the effects a DBD may have upon a NACA0009 airfoil. The goal is to model the data obtained by Corke [7] in Figure 12, Figure 13, Figure 14, and Figure 15. The force density profile chosen was the Roy and Gaitonde profile and its setup parameters are given in Table 9.

Table 9. Roy and Gaitonde Weighting Profile Parameters for Airfoil

$\frac{\text{Power (W)}}{\text{1 meter}}$	Thermal Energy %	Thermal Power $\frac{W}{m}$	Momentum %	Momentum Power $\frac{W}{m}$	X-Momentum		Y-Momentum	
					%	Power $\frac{W}{m}$	%	Power $\frac{W}{m}$
5.0	97.8%	4.89	2.2%	0.11	90%	0.099	10%	0.011

The airfoil was then simulated over 1000 time steps of 0.001 seconds for a total of 1 second simulation time. Up to 20 iterative steps were allowed for each time step for increased accuracy, as was done with the flat plate simulations. The setup of the flow was such that the free stream velocities were 15.2 m/s and 30.4 m/s to yield Re=180k and Re=360k at 0.75c respectively.

As was deduced from the Corke paper [7], the placement of the DBD at 0.75c has not had any appreciable effect on the stall characteristic of the airfoil. Similar to Figure 12 and Figure 14, the two figures generated by the simulation sets, Figure 61 and Figure 62, show a similar minimal effect.

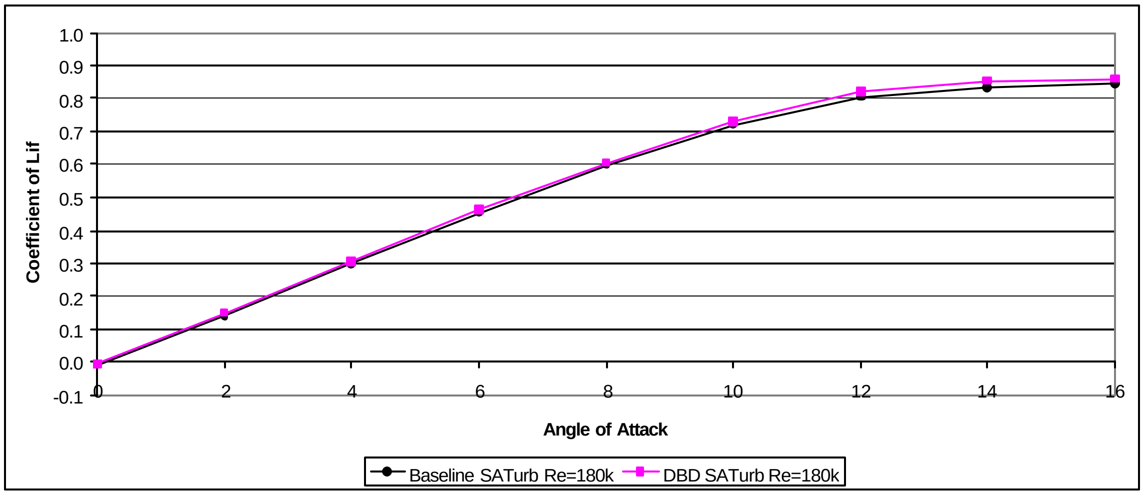


Figure 61
NACA0009 Airfoil C_l vs. AoA for Re=180k

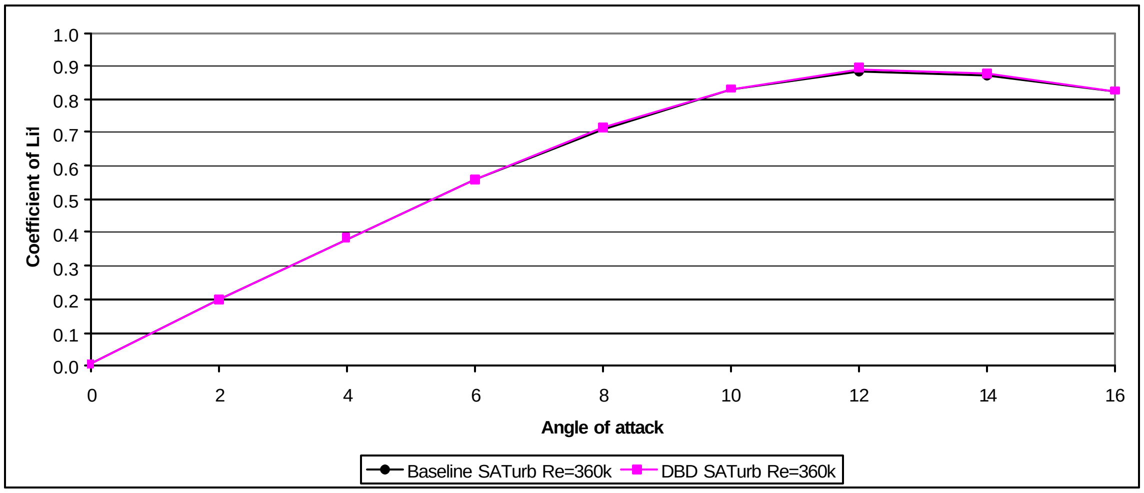


Figure 62
NACA0009 Airfoil C_l vs. AoA for Re=360k

The opposite is the case when comparing Corke's results [7] of Figure 13 and Figure 15 with Figure 63 and Figure 64 respectively. The C_l vs. C_d curves are not as separated and distinct. A closer examination of the data was performed by taking the difference between the simulated data with a DBD in operation and the baseline cases without a DBD in operation.

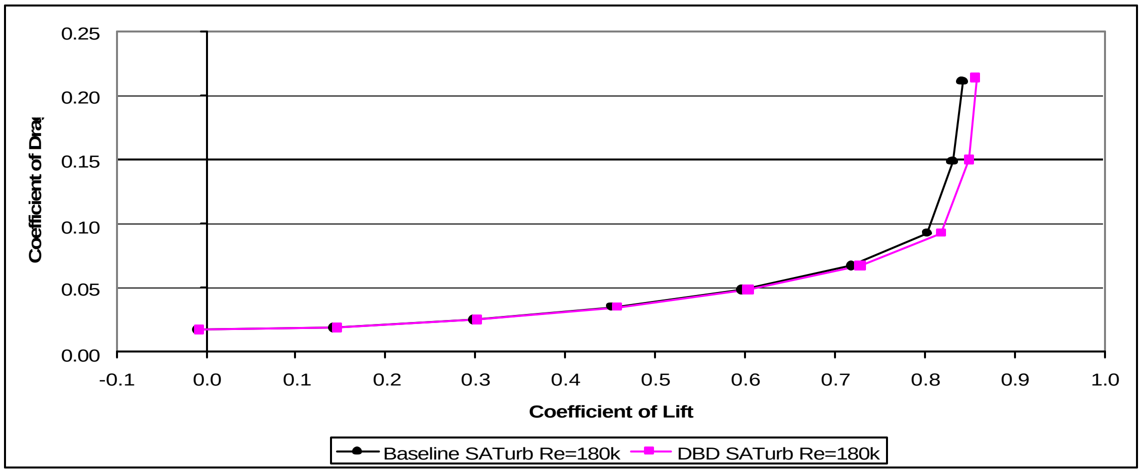


Figure 63
NACA0009 Airfoil C_l vs. C_d for Re=180k

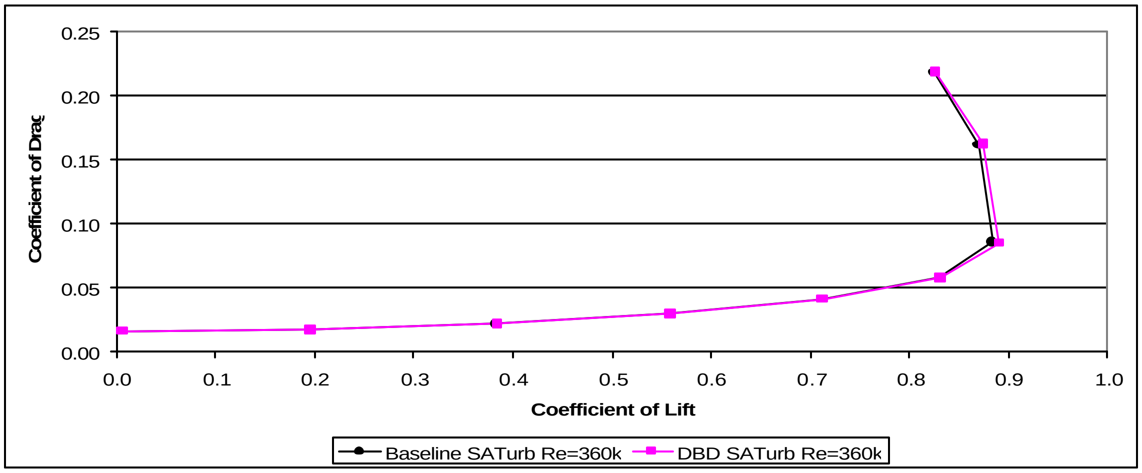


Figure 64
NACA0009 Airfoil Cl vs. Cd for Re=360k

The trend of a positive shift is still seen, but it is very small as shown in Figure 65 and Figure 66. The erratic behavior of the data difference above 12° AoA shows the simulation of the DBD is most likely only valid until this point. Separation of the flow is suspected to have occurred at this angle of attack. Because the DBD is located at 0.75c, the contribution to the airfoil's performance is expected to be minimal. This is due to a minimal amount of separation occurring at about 0.5c at an angle of attack of 0°.

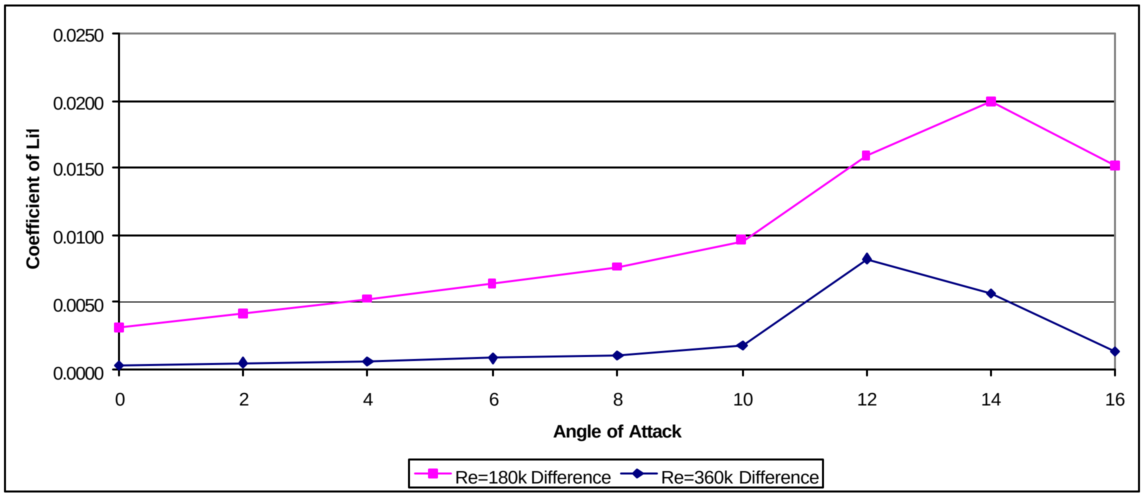


Figure 65
NACA0009 Airfoil C_l Difference for DBD [On-Off] vs. AoA for Re=180k

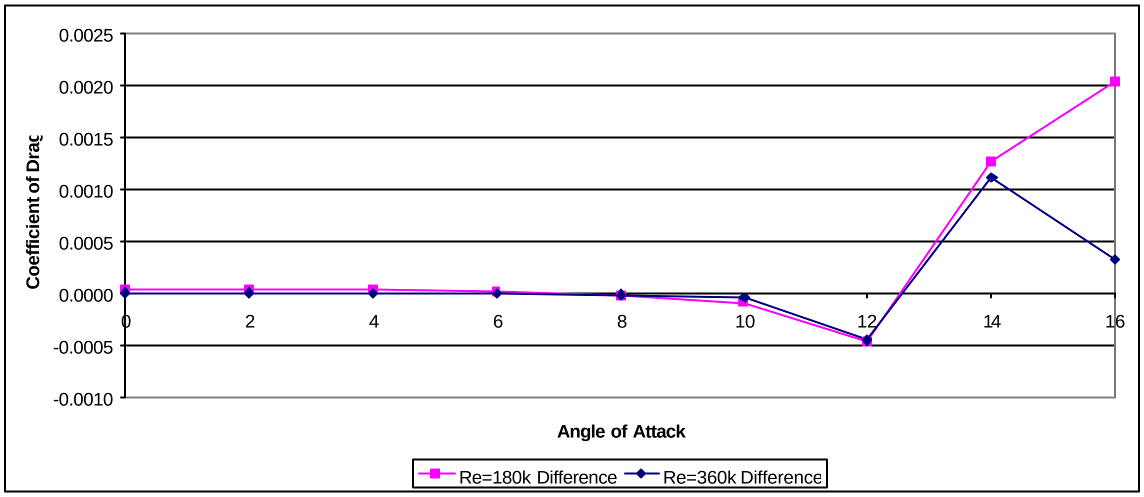


Figure 66
NACA0009 Airfoil C_l Difference for DBD [On-Off] vs. AoA for Re=360k

V. Conclusions

The UDF simulation tool developed has great flexibility for injecting thermal energy and momentum into the flow. Several variables may be tailored, as well as the weighting functions that control the force density and power density for simulation of the DBD induced wall-jet. Fluent®, the commercial software used for these simulations, was fast and accurate with great capability and flexibility. The addition of the UDF source terms to simulate a DBD was much less complex and easier to use than the creation of a special ionized flow solver. A plasma simulation, constrained by time steps of 10's of nanoseconds would take an excessive amount of time and processing power. The code presented mitigates this problem by simplifying the issue to thermal energy and momentum addition into the system using temporal averages of these two sources. The code has the ability to take into account a varying time step that can be smaller or larger than the driving voltage wave's period. Many other parameters, such as the mathematical description of the spatial weighting of cells for simulation of the DBD, the number of DBD elements, and the driving voltage frequency can also be modified. However, extreme care is necessary when implementing the UDF source term to have it operate properly.

Given the two force density profiles of Boeuf and Pitchford and of Roy and Gaitonde, the profile that simulates the momentum addition into the flow the best is the Roy and Gaitonde force density profile. The limited geometry size of the Boeuf and Pitchford force density profile did not allow the ability to overcome the local flow at low velocities, which is not what is seen in experiments. Further, when the horizontal extent of the Boeuf and Pitchford force density profile was increased, minimal change was observed in the downstream velocity profiles. These jets dissipated quickly after the force propelling them was removed. The Roy and Gaitonde force density profile appears to address the momentum addition correctly by having a significant vertical extent as well as horizontal. Also, the thermal addition effect for Roy and Gaitonde is consistent with estimates from section III and simulation results depicted in Figure 25 for temperature increases.

Even though the majority of power is put into the thermal source, it is my conclusion that DBD performance is tied to the momentum source as opposed to the thermal source. This was demonstrated in the results of Figure 57, when the momentum source was fixed at 0.18 W/m input power and the thermal source varied by nearly 4 orders of magnitude from 0.02 W/m to 199.82 W/m with minimal wall jet velocity difference in the cases. Font [19] performed a further study on this effect using an air chemistry model that included both positive and negative ions and came to the same conclusion.

The DBD performance does not have fixed coefficients; that is, the percentage of power going towards thermal addition and momentum addition are not fixed as the power is increased. If this were the case, then the velocity of the wall jet would increase as the voltage, and subsequent power, was increased in experiments. This was not the case as reported by Newcamp [18] in experiments and shown in simulations previously discussed.

The airfoil simulations concluded that Corke's data [7] could be successfully simulated with a Roy and Gaitonde force density profile. The resulting comparison between the Fluent® simulations and Corke's data showed a similar minimal performance improvement trend up to 12° AoA. This capability should lead to faster, lower fidelity, simulations of DBD's on airfoils and low-Re_c turbine blade research.

Suggested follow-on research is to modify for frequency response of the system, vary placement of the DBD on a NACA 0009 airfoil to examine system performance, and to vary the number of DBD devices and examine their effects. The time steps taken were only able to resolve 10 cycles of the 10 kHz driving voltage to an average. To examine the effectiveness of frequency and input waveform on the system as Likhanskii [20] did, the time steps would need to be lowered such that several samples are taken for every period to accurately resolve a temporal solution. For this follow-on effort, limited modification to the thermal energy source and momentum source may be required. DBD placement on the NACA 0009 airfoil will need to compare against experimental data. This data will need to be generated and compared with the follow-on simulations for DBD placement. The final parameter suggested to be modified under follow-on research is the number of DBD's and their spacing. The code is set up to add several areas of weighting for a multiple source DBD simulation at a fixed interval distance. All sources would behave identically with respect to input power to the flow, momentum addition, and thermal energy addition. Effects of increased lift while maintaining drag should be examined and compared with Corke's data [7] as the number of simulated DBD's increases.

The UDF simulation tool developed has great flexibility for injecting thermal energy and momentum into the flow with follow-on research that can be done from the existing code. The creation of the UDF code allowed for examination and macro-analysis of the Boeuf and Pitchford and the Roy and Gaitonde DBD force density profiles on a flat plate geometry. The UDF code then utilized the Roy and Gaitonde force density profile to simulate a DBD on a NACA 0009 airfoil to compare results with Corke's [7] experimental data. These efforts were the purpose of this research.

VI. Appendix A

The tables that follow contain the data points used to create the NACA 0009 airfoil. These points scale to a cord length of 0.202 meters and represent only the top portion of the airfoil. The bottom portion is a mirror image as this is a symmetric airfoil.

Table 10. Computer Generated NACA 0009 Airfoil Data Point Set

X	Y	X	Y	X	Y	X	Y
0.0000000	0.0000000						
0.0000017	0.0000785	0.0000873	0.0005561	0.0069337	0.0045703		
0.0000034	0.0001109	0.0000890	0.0005615	0.0071003	0.0046189	0.0570650	0.0090821
0.0000051	0.0001358	0.0000907	0.0005668	0.0072670	0.0046667	0.0585800	0.0090893
0.0000068	0.0001567	0.0000924	0.0005721	0.0074336	0.0047137	0.0600950	0.0090924
0.0000086	0.0001752	0.0000942	0.0005773	0.0076003	0.0047601	0.0616100	0.0090917
0.0000103	0.0001919	0.0000959	0.0005825	0.0077669	0.0048058	0.0631250	0.0090872
0.0000120	0.0002072	0.0000976	0.0005876	0.0079336	0.0048509	0.0646400	0.0090791
0.0000137	0.0002214	0.0000993	0.0005927	0.0081002	0.0048954	0.0661550	0.0090674
0.0000154	0.0002348	0.0001010	0.0005977	0.0082669	0.0049392	0.0676700	0.0090523
0.0000171	0.0002475	0.0001010	0.0005977	0.0084335	0.0049824	0.0691850	0.0090338
0.0000188	0.0002595	0.0002677	0.0009672	0.0086002	0.0050251	0.0707000	0.0090122
0.0000205	0.0002710	0.0004343	0.0012266	0.0087668	0.0050672	0.0722150	0.0089873
0.0000223	0.0002820	0.0006010	0.0014377	0.0089335	0.0051087	0.0737300	0.0089595
0.0000240	0.0002926	0.0007676	0.0016197	0.0091001	0.0051497	0.0752450	0.0089286
0.0000257	0.0003028	0.0009343	0.0017817	0.0092668	0.0051902	0.0767600	0.0088949
0.0000274	0.0003127	0.0011009	0.0019290	0.0094334	0.0052302	0.0782750	0.0088584
0.0000291	0.0003223	0.0012676	0.0020647	0.0096001	0.0052697	0.0797900	0.0088192
0.0000308	0.0003316	0.0014342	0.0021911	0.0097667	0.0053087	0.0813050	0.0087773
0.0000325	0.0003406	0.0016009	0.0023098	0.0099334	0.0053473	0.0828200	0.0087328
0.0000342	0.0003494	0.0017675	0.0024219	0.0101000	0.0053853	0.0843350	0.0086859
0.0000359	0.0003580	0.0019342	0.0025283	0.0116150	0.0057121	0.0858500	0.0086364
0.0000377	0.0003664	0.0021008	0.0026297	0.0131300	0.0060081	0.0873650	0.0085846
0.0000394	0.0003746	0.0022675	0.0027268	0.0146450	0.0062780	0.0888800	0.0085305
0.0000411	0.0003826	0.0024341	0.0028200	0.0161600	0.0065255	0.0903950	0.0084741
0.0000428	0.0003904	0.0026008	0.0029096	0.0176750	0.0067531	0.0919100	0.0084155
0.0000445	0.0003981	0.0027674	0.0029960	0.0191900	0.0069632	0.0934250	0.0083547
0.0000462	0.0004056	0.0029341	0.0030796	0.0207050	0.0071575	0.0949400	0.0082919
0.0000479	0.0004130	0.0031007	0.0031605	0.0222200	0.0073374	0.0964550	0.0082270
0.0000496	0.0004203	0.0032674	0.0032389	0.0237350	0.0075042	0.0979700	0.0081601
0.0000514	0.0004274	0.0034340	0.0033150	0.0252500	0.0076589	0.0994850	0.0080912
0.0000531	0.0004344	0.0036007	0.0033890	0.0267650	0.0078024	0.1010000	0.0080204
0.0000548	0.0004413	0.0037673	0.0034611	0.0282800	0.0079355	0.1058095	0.0077836
0.0000565	0.0004481	0.0039340	0.0035313	0.0297950	0.0080589	0.1106190	0.0075293
0.0000582	0.0004548	0.0041006	0.0035998	0.0313100	0.0081731	0.1154286	0.0072586
0.0000599	0.0004614	0.0042673	0.0036666	0.0328250	0.0082786	0.1202381	0.0069725
0.0000616	0.0004679	0.0044339	0.0037319	0.0343400	0.0083760	0.1250476	0.0066719
0.0000633	0.0004743	0.0046006	0.0037958	0.0358550	0.0084658	0.1298571	0.0063576
0.0000651	0.0004806	0.0047672	0.0038582	0.0373700	0.0085482	0.1346667	0.0060302
0.0000668	0.0004869	0.0049339	0.0039194	0.0388850	0.0086236	0.1394762	0.0056902
0.0000685	0.0004930	0.0051005	0.0039793	0.0404000	0.0086924	0.1442857	0.0053381
0.0000702	0.0004991	0.0052672	0.0040381	0.0419150	0.0087548	0.1490952	0.0049742
0.0000719	0.0005051	0.0054338	0.0040957	0.0434300	0.0088112	0.1539048	0.0045987
0.0000736	0.0005110	0.0056005	0.0041522	0.0449450	0.0088617	0.1587143	0.0042117
0.0000753	0.0005169	0.0057671	0.0042077	0.0464600	0.0089067	0.1635238	0.0038133
0.0000770	0.0005227	0.0059338	0.0042622	0.0479750	0.0089463	0.1683333	0.0034034
0.0000787	0.0005284	0.0061004	0.0043157	0.0494900	0.0089808	0.1731429	0.0029820
0.0000805	0.0005341	0.0062671	0.0043683	0.0510050	0.0090103	0.1779524	0.0025487
0.0000822	0.0005397	0.0064337	0.0044201	0.0525200	0.0090350	0.1827619	0.0021032
0.0000839	0.0005452	0.0066004	0.0044710	0.0540350	0.0090551	0.1875714	0.0016452
0.0000856	0.0005507	0.0067670	0.0045211	0.0555500	0.0090707	0.1923810	0.0011742
						0.2020000	0.0000000

Table 11. Data Set #1 [12]

x	y
1.00000	0.00000
0.99572	0.00057
0.98296	0.00218
0.96194	0.00463
0.93301	0.00770
0.89668	0.01127
0.85355	0.01522
0.80438	0.01945
0.75000	0.02384
0.69134	0.02823
0.62941	0.03247
0.56526	0.03638
0.50000	0.03978
0.43474	0.04248
0.37059	0.04431
0.33928	0.04484
0.30866	0.04509
0.27886	0.04504
0.25000	0.04466
0.22221	0.04397
0.19562	0.04295
0.17033	0.04161
0.14645	0.03994
0.12408	0.03795
0.10332	0.03564
0.08427	0.03305
0.06699	0.03023
0.05156	0.02720
0.03806	0.02395
0.02653	0.02039
0.01704	0.01646
0.00961	0.01214
0.00428	0.00767
0.00107	0.00349
0.00000	0.00000

Table 12. Data Set #2 [14]

x	y
1.000000	0.000000
0.992588	0.000977
0.978431	0.002727
0.962932	0.004519
0.946984	0.006252
0.930890	0.007916
0.914755	0.009526
0.898598	0.011088
0.882422	0.012606
0.866228	0.014086
0.850019	0.015533
0.833798	0.016950
0.817571	0.018340
0.801341	0.019703
0.785109	0.021040
0.768877	0.022350
0.752645	0.023634
0.736410	0.024889
0.720173	0.026117
0.703932	0.027318
0.687689	0.028491
0.671446	0.029638
0.655203	0.030756
0.638962	0.031844
0.622721	0.032902
0.606482	0.033927
0.590242	0.034920
0.574002	0.035878
0.557763	0.036802
0.541527	0.037689
0.525293	0.038538
0.509062	0.039347
0.492833	0.040113
0.476607	0.040835
0.460383	0.041511
0.444162	0.042138
0.427944	0.042716
0.411732	0.043241
0.395527	0.043709
0.379330	0.044117
0.363143	0.044460
0.346965	0.044734
0.330794	0.044937
0.314630	0.045063
0.298486	0.045109
0.282373	0.045063
0.266276	0.044915
0.250174	0.044663
0.234093	0.044308
0.218057	0.043835
0.202046	0.043231
0.186062	0.042490
0.170138	0.041598
0.154273	0.040535
0.138473	0.039283
0.122778	0.037819
0.107202	0.036110
0.091745	0.034125
0.076471	0.031840
0.061545	0.029225
0.047280	0.026246
0.034392	0.022919
0.023705	0.019346
0.015757	0.015817
0.010145	0.012508
0.006356	0.009616
0.003720	0.007082
0.001851	0.004763
0.000663	0.002678
0.000076	0.000856
0.000000	0.000000

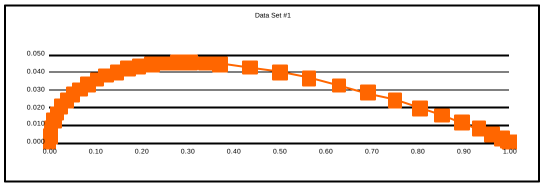


Figure 67
Airfoil Data Set #1 Plot of Table 11

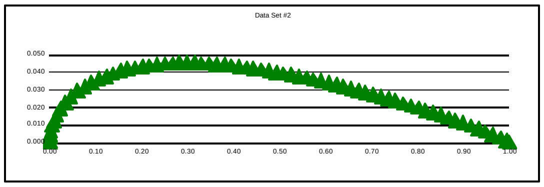


Figure 68
Airfoil Data Set #1 Plot of Table 12

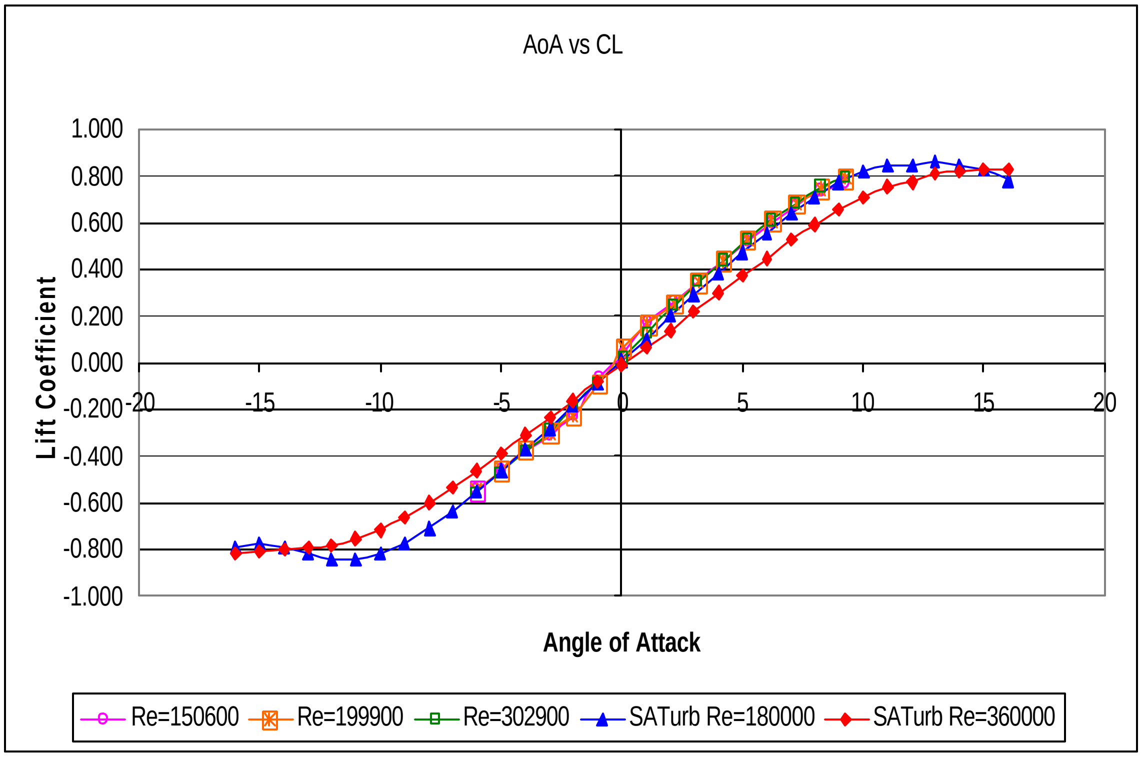


Figure 69
NACA 0009 Angle of Attack VS Lift Coefficient
for Spalart-Allmaras Turbulence Model

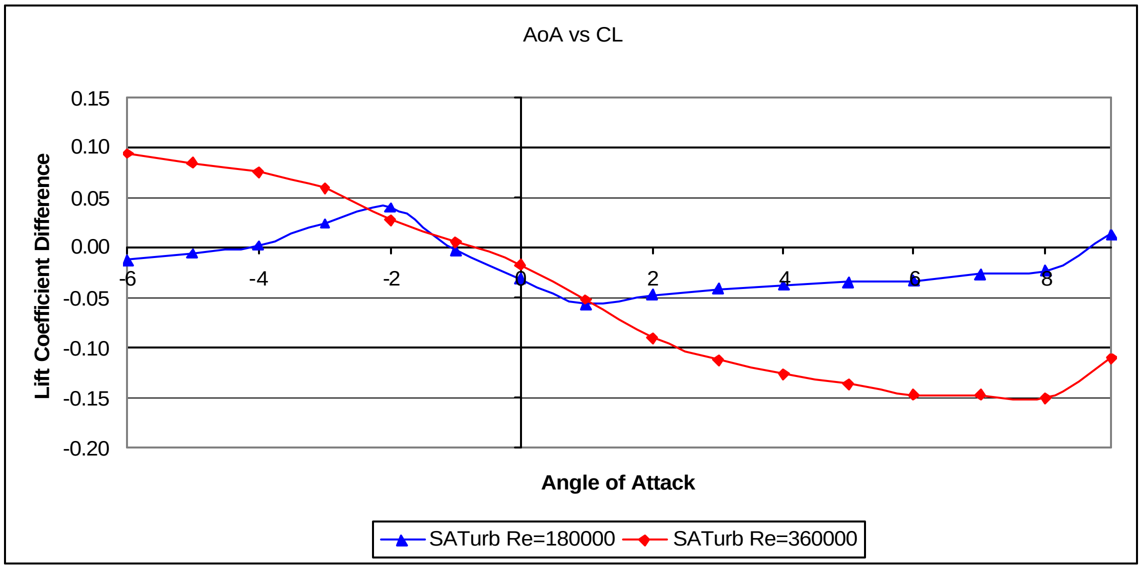


Figure 70
NACA 0009 Angle of Attack VS Lift Coefficient Difference
for Spalart-Allmaras Turbulence Model as Compared to Experimental Data by Selig [15]

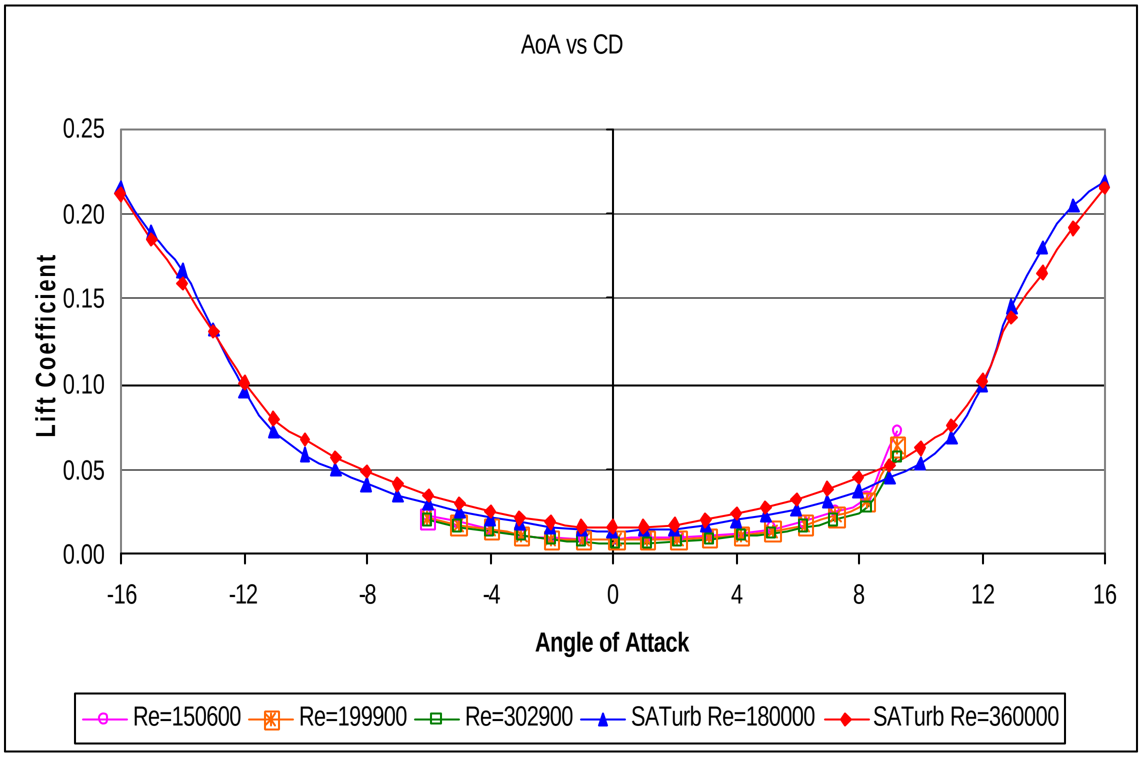


Figure 71
NACA 0009 Lift Coefficient VS Drag Coefficient
for Spalart-Allmaras Turbulence Model

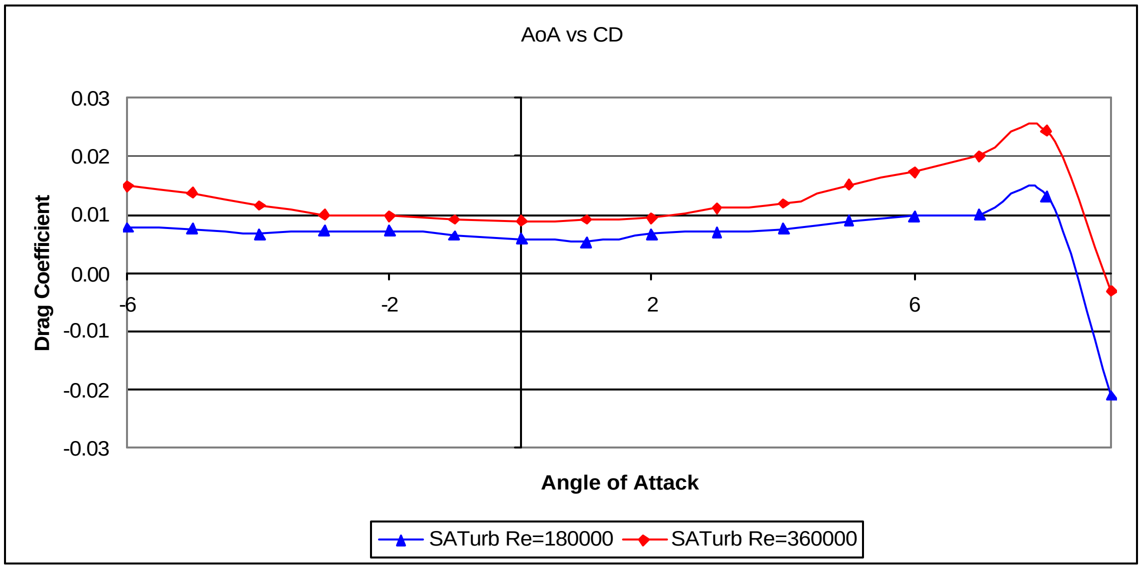


Figure 72
NACA 0009 Lift Coefficient VS Drag Coefficient Difference
for Spalart-Allmaras Turbulence Model as Compared to Experimental Data by Selig [15]

VII. Appendix B

The appendix describes, in detail, the workings of each part of the Fluent® compiled C code and how verification of each piece was performed. The code has several areas that had to be commented to allow compilation by Fluent's® C compiler.

The file that contains all of the following code is "temp_mom_src_trm_FLUENT.c" and has only one header file for its function definitions that Fluent® requires. The code is detailed in the order it is written in the file. The most recent version of the code was modified on 19 August 2005.

Include Files

Originally, the C++ include files were used in this program. However the Fluent® C compiler was unable to identify them, and as a result, the standard C files had to be included instead via the udf.h include file.

```
/* Fluent include files*/
#include "udf.h"
```

Function Definitions

Function definitions are required at the beginning of every C or C++ program to identify the subroutines that are in the following code. The asterisks tell the compiler that a pointer is to be used. This is necessary because several variables are sometimes needed to be modified and returned to the previous subroutine.

Each subroutine will be described in detail as to its inner workings and verification method.

```
/*
**********************************************************************
Subroutine:  Function Definitions

Modified:    04/07/2005
Created:     04/04/2005
Creator:     Capt Timothy R. Klein

Description: Defines Functions to be used throughout file in all
             subroutines.
**********************************************************************
*/

void   read_data  ();

real power_avg  (real,real);
real power_funct(real);

void   coord_xform(real,real, real *,real *, real,real, real,real);

real four_point (real,real, real,real, real,real, real,real, real);
real three_point(real,real, real,real, real,real,            real);

real line_side  (int,real,real);

void   line_intercept(int, real,real, real,real, real *,real *);

real line_offset(int);

real curve_y    (real,real);
real curve_dy   (real,real);

real volume_integration(real,real, real,real, real,real, real);

real weight_funct(real,real);
```

Constant Definitions

The research required that certain parameters be easily changed from one run of data to the next. Therefore, it was necessary to create a global variable set as written below.

```
/*
***********************************************************************
Subroutine:  Constant Definitions

Modified:    08/17/2005
Created:     03/21/2005
Creator:     Capt Timothy R. Klein

Description: Defines Constants to be used throughout file in all
             subroutines.
***********************************************************************
*/

/* Boltzmann constant, Joules/Kelvin */
#define kb             1.3807E-23

/* Total Power in Watts per Unit Length (1.0 meter) */
#define POWER_TOT       80.0

/* TEMP_PERCENT + MOM_PERCENT = 100.0% */
#define TEMP_PERCENT    98.0
#define  MOM_PERCENT     2.0

/* X_MOM + Y_MOM = 100.0% of MOM_PERCENT */
#define X_MOM           99.0
#define Y_MOM            1.0

/* Number of DBD's*/
#define DBD_NUM          1

/* Distance in meters between DBD's if multiple */
#define DBD_SPACING      0.050

/* Physical Location of first DBD (Source)
   If multiple DBD's, then X_POSITION*CORD + n*DBD_SPACING
     Where n is the number of the DBD starting at 0
   Distance based on a 1 meter cord length */
#define X_POSITION       0.75  /* cord */
#define Y_POSITION       0.0   /* cord */

/* Frequency of DBD Voltage in Hz */
#define FREQ             10000.0
```

```
/* Cord Length (Flat Plate or Airfoil) */
#define CORD             0.202  /* meter */

/* Airfoil Span for 3D Calculations*/
#define SPAN             1.0    /* meter */

/* Switch for Flat-Plate or Airfoil
   Flat-Plate: F_OR_A = 0
   Airfoil:    F_OR_A = 1 */
#define F_OR_A           0

/* Switch for Boueff&Pitchford or Roy&Gaitonde Weighting-Scheme
   Boueff&Pitchford: WScheme = 1
   Roy&Gaitonde:     WScheme = 2 */
#define WScheme          2
```

Cell Weighting Define on Demand UDF Code

This section of code defines the Cell Weighting Define on Demand UDF that was used in Fluent®.

```
/*
***********************************************************************
Subroutine:  DEFINE_ON_DEMAND(cell_weight_on_demand)

Modified:    08/08/2005
Created:     03/21/2005
Creator:     Capt Timothy R. Klein

Description: Loops over all cells to weight each one.  Total Area is
             found and then normalized by the Total Area to give 1.
***********************************************************************
*/
DEFINE_ON_DEMAND(cell_weight_on_demand)
{
  /* Define Fluent cell variables */
  Domain *d;
  Thread *t;
  cell_t c;
  Node   *node;

  d = Get_Domain(1);
  t = Lookup_Thread(d, 2);  /* Get fluid thread using Fluent utility */
                            /*   Zone=2 is for the fluid             */

  real  x[4], y[4];
  real  u[4], v[4];

  int tot_nodes, count;
  int i, n;

  real tot_weight = 0.0;  /* Set initial Total Weight        */
  real weight     = 0.0;  /* Set initial Weight for the cell */

  if (N_UDM<(DBD_NUM+3+1))
  {
    printf("\n\n\nYOU MUST DEFINE %d UDFM's!!\n",(DBD_NUM+3+1));
    Internal_Error("YOU MUST DEFINE more UDFM's!!\n\n\n");
  }
```

```
/* Fill the UDM (User Defined Memory) with cell weight */
thread_loop_c(t,d)
{

  /* Loop for Multiple DBD's */
  for(i=1;i<=DBD_NUM;++i)
  {
    /* Set Location of DBD and find Parallel Vector for Coordinate
       Transformations */
    real px = CORD*X_POSITION + ((real)(i-1)*DBD_SPACING);
    real py = curve_y(px,CORD);
    real vx = 1.0;
    real vy = curve_dy(px,CORD);
    real vmag = sqrt(vx*vx+vy*vy);
    vx /= vmag;
    vy /= vmag;

    /* Weight each cell */
    begin_c_loop(c,t)  /* Cell Loop */
    {
      C_UDMI(c,t,i) = 0.0;

      /* Get the total number of nodes in a cell    */
      tot_nodes = C_NNODES(c,t);
      /* Get the coordinates of the nodes in a cell */
      c_node_loop(c,t,count)  /* Node Loop */
      {
        node = C_NODE(c,t,count);
        x[count] = NODE_X(node);
        y[count] = NODE_Y(node);
      }
      count = 0;

      weight = 0.0;  /* Set initial Weight for the cell */

      for(count=0;count<tot_nodes;++count)
      {
        coord_xform(x[count],y[count], &u[count],&v[count], vx,vy,
                    px,py);
      }
```

```
        /* Send points to functions to see if a cell straddles a line
           and to give weight */
        if(tot_nodes == 4)
        {
          weight = four_point (u[0],v[0], u[1],v[1], u[2],v[2],
                               u[3],v[3], weight);
        }
        else if(tot_nodes == 3)
        {
          weight = three_point(u[0],v[0], u[1],v[1], u[2],v[2],
                               weight);
        }
        else
        {
          printf("ERROR[cell_weight_on_demand]: Total Number of Nodes
                  NOT 3 or 4!\n");
        }

        C_UDMI(c,t,i) = weight;
        tot_weight += fabs(weight);

        if(c%(100*100)==0) printf("\n");/*New line every 100 "."*/
        if(c%100      ==0) printf("."); /*Shows progress of cell loop*/
      }
      end_c_loop(c,t)    /* Cell Loop */

      /* Normalize each cell */
      begin_c_loop(c,t)  /* Cell Loop */
      {
        C_UDMI(c,t,i) /= tot_weight;    /*Normalize each cell*/
        if(c%(100*100)==0) printf("\n");/*New line every 100 "."*/
        if(c%100      ==0) printf("."); /*Shows progress of cell loop*/
      }
      end_c_loop(c,t)    /* Cell Loop */

    }  /* DBD_NUM for loop */

  }  /* Thread Loop */

}  /* DEFINE_ON_DEMAND(cell_weight_on_demand) */
```

Temperature Source UDF Code

This section of code defines the Temperature Source UDF that was used in Fluent®.

```
/*
***********************************************************************
Subroutine:  DEFINE_SOURCE(temp_source)

Modified:    08/19/2005
Created:     03/21/2005
Creator:     Capt Timothy R. Klein

Description: Adds thermal energy to cell given the cell, power, and
             weighting.
***********************************************************************
*/
DEFINE_SOURCE(temp_source,c,t,dS,eqn)
{

  if(TEMP_PERCENT>0.0)  /* Less Computational Time */
  {
    real weight = 0.0;
    real source;

    int i;

    /* Get normalized cell weight from memory */
    /* i represents each DBD if multiple      */

    for(i=0;i<DBD_NUM;++i)
    {
      weight += C_UDMI(c,t,i+1);
    }

    if(weight != 0.0)
    {
      real volume  = C_VOLUME(c,t);

      /* This needs to be a volume ratio of W/m^3 */
      source = POWER_TOT*(TEMP_PERCENT/100.0)*fabs(weight)/volume;
      C_UDMI(c,t,DBD_NUM+3) = source; /*if DBD_NUM=1 then UDMI slot=4*/
    }
    else
    {
      source = 0.0;
      C_UDMI(c,t,DBD_NUM+3) = source; /*if DBD_NUM=1 then UDMI slot=4*/
    }
    dS[eqn] = 0.0;
    return source;

  }                  /* TEMP_PERCENT >  0.0 */
  else               /* TEMP_PERCENT <= 0.0 */
  {
    C_UDMI(c,t,DBD_NUM+3) = 0.0; /*if DBD_NUM=1 then UDMI slot=4*/
    dS[eqn] = 0.0;
    return    0.0;
  }

}  /* DEFINE_SOURCE(temp_source) subroutine end */
```

X-Momentum Source UDF Code

This section of code defines the X-Momentum Source UDF that was used in Fluent®.

```
/*
***********************************************************************
Subroutine:  DEFINE_SOURCE(x_momentum_source)

Modified:    08/19/2005
Created:     03/21/2005
Creator:     Capt Timothy R. Klein

Description: Adds x-momentum energy to cell given the cell, power, and
             weighting.
***********************************************************************
*/
DEFINE_SOURCE(x_momentum_source,c,t,dS,eqn)
{
  if(MOM_PERCENT>0.0)  /* Less Computational Time */
  {
    real weight = 0.0;
    real source;
    real u02, v02;

    int i;
    /* Get normalized cell weight from memory */
    /* i represents each DBD if multiple      */
    for(i=0;i<DBD_NUM;++i)
    {
      weight += C_UDMI(c,t,i+1);
    }

    real dt = CURRENT_TIMESTEP;   /* Get Current Timestep */

    if(weight != 0.0)
    {
      real energy = POWER_TOT*dt;

      /* Calculate weighted energy into cell */
      real energy_x = (MOM_PERCENT/100.0)*(X_MOM/100.0)*weight*energy;
      real energy_y = (MOM_PERCENT/100.0)*(Y_MOM/100.0)*weight*energy;

      /* Calculate mass in cell */
      real density = C_R(c,t);
      real volume  = C_VOLUME(c,t);
      real mass    = volume*density;

      /* Get Initial Cell Velocities and Kinetic Energy */
      real u00 = C_U(c,t);  /* initial u velocity       */
      real v00 = C_V(c,t);  /* initial v velocity       */
```

```
/* ************************************************************/
/* Allow for direction of source to be along surface tangent   */

/* Set Surface Tangent Vector at DBD Location                  */
real px = CORD*X_POSITION + ((real)(i)*DBD_SPACING);
real vx = 1.0;
real vy = curve_dy(px,CORD);


/* Magnitude of vector v                                       */
real v_xy_mag  = sqrt(vx*vx + vy*vy);
vx /= v_xy_mag;
vy /= v_xy_mag;

/*Reorient x&y velocity vectors to be tangent to surface at DBD*/
real u01 =  u00*vx + v00*vy;
real v01 = -u00*vy + v00*vx;

/* Calculate Cell Velocities and Kinetic Energy after energy
   addition */
real KE1x = energy_x;
real KE1y = energy_y;

/* Need to preserve Energy direction by keeping velocity sign  */
/*  Avoids sqrt of negative number and allows for deceleration */
/*  if flow moving in opposite direction of DBD wall jet       */
real u02sqr = u01*fabs(u01) + (2*(KE1x)/mass);
if(u02sqr < 0.0)
{
  u02 = - sqrt( fabs( u02sqr ) );
}
else
{
  u02 =   sqrt(       u02sqr   );
}

/* Need to preserve Energy direction by keeping velocity sign  */
/*  Avoids sqrt of negative number and allows for deceleration */
/*  if flow moving in opposite direction of DBD wall jet       */
real v02sqr = v01*fabs(v01) + (2*(KE1y)/mass);
if(v02sqr < 0.0)
{
  v02 = - sqrt( fabs( v02sqr ) );
}
else
{
  v02 =   sqrt(       v02sqr   );
}

/* Return to x&y coordinate system */
real u11 =  u02*vx - v02*vy;
real acl = (u11-u00)/dt;

/* *********************************************************** */
```

```
      /* Density instead of mass so there is a volume ratio     */
      /* momentum source = N/m^3 = density * acceleration       */
      source = density*acl;                            /* Debug */
      C_UDMI(c,t,DBD_NUM+1) = source; /*if DBD_NUM=1 then UDMI slot=2*/
    }
    else
    {
      source = 0.0;
      C_UDMI(c,t,DBD_NUM+1) = source; /*if DBD_NUM=1 then UDMI slot=2*/
    }

    dS[eqn] = 0.0;
    return source;

  }                 /* MOM_PERCENT >  0.0 */
  else              /* MOM_PERCENT <= 0.0 */
  {
    C_UDMI(c,t,DBD_NUM+1) = 0.0; /*if DBD_NUM=1 then UDMI slot=2*/
    dS[eqn] = 0.0;
    return    0.0;
  }

}  /* DEFINE_SOURCE(x_momentum_source) subroutine end */
```

Y-Momentum Source UDF Code

This section of code defines the Y-Momentum Source UDF that was used in Fluent®.

```
/*
***********************************************************************
Subroutine:  DEFINE_SOURCE(y_momentum_source)

Modified:    08/16/2005
Created:     03/21/2005
Creator:     Capt Timothy R. Klein

Description: Adds y-momentum energy to cell given the cell, power, and
             weighting.
***********************************************************************
*/
DEFINE_SOURCE(y_momentum_source,c,t,dS,eqn)
{
  if(MOM_PERCENT>0.0)  /* Less Computational Time */
  {
    real source, u02, v02, weight = 0.0;

    int i;
    /* Get normalized cell weight from memory */
    /* i represents each DBD if multiple      */
    for(i=0;i<DBD_NUM;++i)
    {
      weight = C_UDMI(c,t,i+1);
    }

    real dt = CURRENT_TIMESTEP;   /* Get Current Timestep */

    if(weight != 0.0)
    {
      real energy = POWER_TOT*dt;

      /* Keep the wall-jet pulled to the surface */
      weight = - fabs(weight);

      /* Calculate weighted energy into cell */
      real energy_x = (MOM_PERCENT/100.0)*(X_MOM/100.0)*weight*energy;
      real energy_y = (MOM_PERCENT/100.0)*(Y_MOM/100.0)*weight*energy;

      /* Calculate mass in cell */
      real density = C_R(c,t);
      real volume  = C_VOLUME(c,t);
      real mass    = volume*density;

      /* Get Initial Cell Velocities and Kinetic Energy */
      real u00 = C_U(c,t);  /* initial u velocity       */
      real v00 = C_V(c,t);  /* initial v velocity       */
```

```
/* ***************************************************************/
/* Allow for direction of source to be along surface tangent   */

/* Set Surface Tangent Vector at DBD Location                  */
real px = CORD*X_POSITION + ((real)(i)*DBD_SPACING);
real vx = 1.0;
real vy = curve_dy(px,CORD);


/* Magnitude of vector v                                       */
real v_xy_mag  = sqrt(vx*vx + vy*vy);
vx /= v_xy_mag;
vy /= v_xy_mag;

/*Reorient x&y velocity vectors to be tangent to surface at DBD*/
real u01 =  u00*vx + v00*vy;
real v01 = -u00*vy + v00*vx;

/* Calculate Cell Velocities and Kinetic Energy after energy
   addition */
real KE1x = energy_x;
real KE1y = energy_y;

/* Need to preserve Energy direction by keeping velocity sign  */
/*  Avoids sqrt of negative number and allows for deceleration */
/*  if flow moving in opposite direction of DBD wall jet       */
real u02sqr = u01*fabs(u01) + (2*(KE1x)/mass);
if(u02sqr < 0.0)
{
  u02 = - sqrt( fabs( u02sqr ) );
}
else
{
  u02 =   sqrt(       u02sqr   );
}

/* Need to preserve Energy direction by keeping velocity sign  */
/*  Avoids sqrt of negative number and allows for deceleration */
/*  if flow moving in opposite direction of DBD wall jet       */
real v02sqr = v01*fabs(v01) + (2*(KE1y)/mass);
if(v02sqr < 0.0)
{
  v02 = - sqrt( fabs( v02sqr ) );
}
else
{
  v02 =   sqrt(       v02sqr   );
}

/* Return to x&y coordinate system */
real v11 = u02*vy + v02*vx;
real acl = (v11-v00)/dt;

/* ***************************************************************/
```

```
        /* Density instead of mass so there is a volume ratio         */
        /* momentum = mass * acceleration                              */
        /* This requires momentum_per_volume = density * acceleration */
        /*   Density is: mass/volume                                   */
        source = density*acl;                          /* Debug */
        C_UDMI(c,t,DBD_NUM+2) = source; /*if DBD_NUM=1 then UDMI slot=3*/
      }
      else
      {
        source = 0.0;
        C_UDMI(c,t,DBD_NUM+2) = source; /*if DBD_NUM=1 then UDMI slot=3*/
      }

      dS[eqn] = 0.0;
      return source;

    }                 /* MOM_PERCENT >  0.0 */
    else              /* MOM_PERCENT <= 0.0 */
    {
      C_UDMI(c,t,DBD_NUM+2) = 0.0; /*if DBD_NUM=1 then UDMI slot=3*/
      dS[eqn] = 0.0;
      return    0.0;
    }

  }  /* DEFINE_SOURCE(y_momentum_source) subroutine end */
```

Main Testing Program for Verification

After all of the parts to this code were finished, it was necessary to check for their correctness of content. The main program that is listed below does this testing, and is commented out for the period of time when the code is inserted into Fluent®.

```
/*
***********************************************************************
Subroutine:  main()

Modified:    04/08/2005
Created:     04/04/2005
Creator:     Capt Timothy R. Klein

Description: Tests different areas of temp_mom_src_trm.cpp for compile-
             ability and correctness of function/subroutine algorithms.
***********************************************************************
*/
int main(int argc, char* argv[])
{
```

Testing of the settings is an easy way to make sure that you are getting all of the correct variables input.

```
  // Test Settings
  std::cout << "Test Settings\n\n";
  std::cout
      << "POWER_TOT   : " << "    20.0   " << POWER_TOT    << "\n"
      << "TEMP_PERCENT: " << "   100.0   " << TEMP_PERCENT << "\n"
      << "MOM_PERCENT : " << "     0.0   " << MOM_PERCENT  << "\n"
      << "X_MOM       : " << "    90.0   " << X_MOM        << "\n"
      << "Y_MOM       : " << "    10.0   " << Y_MOM        << "\n"
      << "DBD_NUM     : " << "     1     " << DBD_NUM      << "\n"
      << "DBD_SPACING : " << "     0.050 " << DBD_SPACING  << "\n"
      << "X_POSITION  : " << "     0.75  " << X_POSITION   << "\n"
      << "Y_POSITION  : " << "     0.0   " << Y_POSITION   << "\n"
      << "FREQ        : " << " 10000.0   " << FREQ         << "\n"
      << "CORD        : " << "     0.202 " << CORD         << "\n"
      << "SPAN        : " << "     1.0   " << SPAN         << "\n"
      << "F_OR_A      : " << "     0     " << F_OR_A       << "\n";
  std::cout << "\n\n\n";
```

The power average and power function subroutines integrate the area under the curve of the power function. Thus, it was tested that between t=0 and t=π radians that the function would return the same value as between t=2π and t=3π. Other areas were compared to ensure accuracy of the integration under the curve.

```
  // Test subroutine: power_avg & power_funct
  std::cout << "Test subroutine: power_avg & power_funct\n\n";
  real t1 = 0.0;
  real t2 = 0.5/FREQ;
  real t3 = 1.0/FREQ;
  real t4 = 1.5/FREQ;
  real t5 = 2.0/FREQ;
  real t6 = 0.3/FREQ;
  real t7 = 1.3/FREQ;
  std::cout << power_avg(t1,t2) << "  " << power_avg(t2,t3) << "\n"
            << power_avg(t3,t4) << "  " << power_avg(t4,t5) << "\n"
            << power_avg(t1,t3) << "  " << power_avg(t2,t4) << "  "
            << power_avg(t3,t5) << "\n"
            // should be the same as power_avg([t1,t3],[t2,t4],[t3,t5])
            << power_avg(t6,t7) << "\n"
            << power_avg(t1,t5) << "\n";
  std::cout << "\n\n\n";
```

Hand calculations were performed for a set of points in free space being rotated to a new coordinate system by a given vector. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the data expected.

```
  // Test subroutine: coord_xform
  std::cout << "Test subroutine: coord_xform\n\n";
  real u,v;
  coord_xform(+3.0,-2.0, u,v, +2.0,+1.0, +2.0,+2.0);
  std::cout << "(2.68328,-3.1305)\n";
  std::cout << "(" << u << "," << v << ")\n";
  std::cout << "\n\n\n";
```

Hand calculations were performed for a set of points in free space on either side of the following lines. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the data expected.

```
  // Test subroutine: line_offset
  std::cout << "Test subroutine: line_offset\n\n";
  std::cout << "Last Column should be 0.000\n"
       << " 0      " << line_offset(1) <<  0.000-line_offset(1) << "\n"
       << "-0.001 " << line_offset(2) << -0.001-line_offset(2) << "\n"
       << " 0.01   " << line_offset(3) <<  0.010-line_offset(3) << "\n"
       << " 0      " << line_offset(4) <<  0.000-line_offset(4) << "\n"
       << " 0.01   " << line_offset(5) <<  0.010-line_offset(5) << "\n";
  std::cout << "\n\n\n"
```

Hand calculations were performed for the point (0.005,0.005). The subroutine line_side returns a double or real value of the distance from the line number the point is compared against. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the data expected.

```
  // Test subroutine: line_side
  std::cout << "Test subroutine: line_side\n\n";
  std::cout << line_side(1,+0.005,+0.005) << "  "
            << line_side(2,+0.005,+0.005) << "  "
            << line_side(3,+0.005,+0.005) << "  "
            << line_side(4,+0.005,+0.005) << "  "
            << line_side(5,+0.005,+0.005) << "\n";
  std::cout << "\n\n\n";
```

Hand calculations were performed for the line created by connecting the points (0.002,0.003) and (-0.002,0.001) across line number 1, where x=0.0. The returned values of x and y are the location at which the point-line intersects line number 1. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the exact data expected.
The case where the points do not sit on either side of the line was also performed. An error message was built in and successfully tested if this condition should occur.

```
  // Test subroutine: line_intercept
  std::cout << "Test subroutine: line_intercept\n\n";
  real x,y;
  line_intercept(1, +0.002,+0.003, -0.002,+0.001, x,y);
  std::cout << "(0,0.002)\n";
  std::cout << "(" << x << "," << y << ")\n\n";
  line_intercept(2, +0.002,+0.003, +0.002,+0.001, x,y);
  std::cout << "Point does not stradle line, therefore (0,0)\n";
  std::cout << "(" << x << "," << y << ")\n\n";
  std::cout << "\n\n\n";
```

Hand calculations were performed for several points. The subroutine weight_funct returns a double or real value from the equation specified at the point given. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the data expected.

```
  // Test subroutine: weight_funct
  std::cout << "Test subroutine: weight_funct\n\n";
  real weight;
  weight = weight_funct( 0.000 , 0.000);
  std::cout << "E: 0\n"
            << "7 " << weight << "\n";
  weight = weight_funct(-0.002 ,-0.001);  // Outside boundary area
  std::cout << "0 " << weight << "\n";
  weight = weight_funct(-0.002 ,+0.001);  // Outside boundary area
  std::cout << "0 " << weight << "\n";
  weight = weight_funct(-0.002 ,+0.011);  // Outside boundary area
  std::cout << "0 " << weight << "\n";
  weight = weight_funct(+0.011 ,-0.001);  // Outside boundary area
  std::cout << "0 " << weight << "\n";
  weight = weight_funct(+0.011 ,+0.001);  // Outside boundary area
  std::cout << "0 " << weight << "\n";
  weight = weight_funct(+0.011 ,+0.011);  // Outside boundary area
  std::cout << "0 " << weight << "\n";
  weight = weight_funct(+0.001 ,+0.011);  // Outside boundary area
  std::cout << "0 " << weight << "\n";
  weight = weight_funct(-0.0005,+0.011);  // Outside boundary area
  std::cout << "0 " << weight << "\n";
  weight = weight_funct(-0.0005,-0.001);  // Outside boundary area
  std::cout << "E: -0.015\n"
            << "0.98511939603 " << weight << "\n";
  weight = weight_funct(-0.0005,+0.001);
  std::cout << "E: -0.015\n"
            << "0.98511939603 " << weight << "\n";
  weight = weight_funct(+0.001 ,-0.001);
  std::cout << "E: -0.002\n"
            << "6.98601399067 " << weight << "\n";
  weight = weight_funct(+0.001 ,+0.001);
  std::cout << "E: -0.002\n"
            << "6.98601399067 " << weight << "\n";
  std::cout << "\n\n\n";
```

Hand calculations were performed for a set of 3 points. The subroutine returned the weight of the cell obtained via descritized integration. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the data to within 0.1% of the expected value. Because this type of integration is not exact and the volume step size was set low for speed, this is a reasonable answer.

```
  // Test subroutine: volume_integration
  std::cout << "Test subroutine: volume_integration\n\n";
  real weight;
  weight =
      volume_integration(0.001,0.004, 0.005,0.008, 0.007,0.001, 0.0);
  std::cout << "0.000125539  " << weight << "\n";
  std::cout << "\n\n\n";
```

Hand calculations were performed for several points. The subroutine returned the y-coordinate of the airfoil curve, given the airfoil is symmetric. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the data to within 10^{-6}% of the expected value; well within acceptable limits.

```
  // Test subroutine: curve_y
  std::cout << "Test subroutine: curve_y\n\n";
  real x;
  x = 0.000;
  std::cout << "(0,0)\n"
            << "(" << x << "," << curve_y(x,CORD) << ")\n\n";
  x = 0.0101;
  std::cout << "(0.0101,0.00538530)\n"
            << "(" << x << "," << curve_y(x,CORD) << ")\n\n";
  x = 0.0707;
  std::cout << "(0.0707,0.00901220)\n"
            << "(" << x << "," << curve_y(x,CORD) << ")\n\n";
  x = 0.101;
  std::cout << "(0.101,0.00802040)\n"
            << "(" << x << "," << curve_y(x,CORD) << ")\n\n";
  x = 0.202;
  std::cout << "(0.202,~0)\n"
            << "(" << x << "," << curve_y(x,CORD) << ")\n";
  std::cout << "\n\n\n";
```

Hand calculations were performed for several points. The subroutine returned the slope of the airfoil curve at the x-coordinate, given the airfoil is symmetric. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the data to within 0.1% of the expected value; well within acceptable limits.

```
  // Test subroutine: curve_dy
  std::cout << "Test subroutine: curve_dy\n\n";
  real x,slope_data;
  x = 0.0000000000000001;
  std::cout << "As x->0.0, then dy->infinity\n";
  std::cout << "(1e-016,infinity)\n"
            << "(" << x << "," << curve_dy(x,CORD) << ")\n\n";
  x = 0.060095;
  slope_data = (1./3.)*( (0.0090893-0.0090924)/(0.058580-0.060095) +
                         (0.0090924-0.0090917)/(0.060095-0.061610) +
                         (0.0090893-0.0090917)/(0.058580-0.061610)  );
  std::cout << "As y->max, then dy->0.0\n";
  std::cout
  << "(0.060095,~0)\n"
  << "(0.060095," << slope_data << ")\n"
  << "(0.060095," << (0.0090893-0.0090924)/(0.058580-0.060095) << ")\n"
  << "(0.060095," << (0.0090924-0.0090917)/(0.060095-0.061610) << ")\n"
  << "(0.060095," << (0.0090893-0.0090917)/(0.058580-0.061610) << ")\n"
  << "(" << x << "," << curve_dy(x,CORD) << ")\n\n";
  x = 0.0707;
  slope_data = (1./3.)*( (0.0090338-0.0090122)/(0.069185-0.070700) +
                         (0.0090122-0.0089873)/(0.070700-0.072215) +
                         (0.0090338-0.0089873)/(0.069185-0.072215)  );
  std::cout << "Use Data around x=0.0707 to get slope and compare\n";
  std::cout << "(0.0707," << slope_data << ")\n"
            << "(" << x << "," << curve_dy(x,CORD) << ")\n\n";
  std::cout << "\n\n\n";
```

Hand calculations were performed for two sets of 3 points. The subroutine returned the weight of the area contained within these 3 points. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the data to within 0.1% of the expected value; well within acceptable limits.

```
  // Test subroutine: three_point
  std::cout << "Test subroutine: three_point\n\n";
  real weight;
  std::cout << "Test #1: All 3 points in Quadrant I\n";
  weight = three_point( 0.001,0.004, 0.005,0.008, 0.007,0.001, 0.0);
  std::cout << "0.000125539  " << weight << "\n\n";
  std::cout
      << "Test #2: 2 points in Quadrant I, 1 point in Quadrant III\n";
  weight = three_point(-0.001,0.004, 0.005,0.008, 0.007,0.001, 0.0);
  std::cout << "0.000172459176  " << weight << "\n\n";
  std::cout << "\n\n\n";
```

Hand calculations were performed for two sets of 4 points. The subroutine returned the weight of the area contained within these 4 points. These calculations were coded below and the subroutine was sent the data to give its results. The program generated the data to within 0.1% of the expected value; well within acceptable limits.

```
  // Test subroutine: four_point
  std::cout << "Test subroutine: four_point\n\n";
  real weight;
  std::cout << "Test #1: All 4 points in Quadrant I\n";
  weight =
  four_point( 0.001,0.004, 0.005,0.008, 0.007,0.001, 0.008,0.009, 0.0);
  std::cout << "0.0002063887  " << weight << "\n\n";
  std::cout
      << "Test #2: 2 points in Quadrant I, 1 point in Quadrant III\n";
  weight =
  four_point(-0.001,0.004, 0.005,0.008, 0.007,0.001, 0.008,0.009, 0.0);
  std::cout << "0.000253308876  " << weight << "\n\n";
  std::cout << "\n\n\n";
```

The following piece of code was written for nothing more than to be able to view the output prior to the main test program completing. Nothing is done with the variable "something".

```
  // Pause to look at output data
  int something;
  std::cin >> something;

  return 0;

}  // main subroutine end
```

Subroutine: power_avg

The subroutine power_avg integrates the area under the curve via a discrete trapezoidal integration scheme. The function integrates the area underneath the curve defined in the power_funct subroutine from time t=t1 to t=t2. The area is then returned by the subroutine. This subroutine was not used in the final program, but is included here for follow-on work where temporal simulations may require it to be run.

```
/*
*************************************************************************
Subroutine:  power_avg

Modified:    03/29/2005
Created:     03/21/2005
Creator:     Capt Timothy R. Klein

Description: Integrates the area under the curve from time t1 to time
             t2.
*************************************************************************
*/

real power_avg(real t1,real t2)
{
  /* Declare Local Variables */
  real avg = 0.0;

  real num_steps = 100.0;
  real dt = 1.0/(num_steps*FREQ);  /* Seconds */

  real t;

  for(t=t1;t<t2;t+=dt)
  {
    /* Trapezoidal Numerical Integration */
    avg += dt*(0.5*( power_funct(t)+power_funct(t+dt) ));
  }

  return avg;

}  /* power_avg subroutine end */
```

Subroutine: power_funct

The subroutine power_funct returns the value of a weighted sine function given a time passed to it. The subroutine uses a function that indicates a pull, giving amplitude of -1 for the first part of the wave. The second part of the wave indicates a push, giving amplitude of -7. This is consistent with Font [8], where there is 7 times more push than pull during a cycle containing one full waveform. This subroutine was not used in the final program, but is included here for follow-on work where temporal simulations may require it to be run.

```
/*
***********************************************************************
Subroutine:  power_funct

Modified:    03/29/2005
Created:     03/21/2005
Creator:     Capt Timothy R. Klein

Description: Power delivery function represented as a sine wave of
             varying peaks.
***********************************************************************
*/

real power_funct(real t)
{
  real const pi = 3.14159;  /* Pi, constant                       */

  real SINE_K1, SINE_K2;    /* Sine wave amplitudes               */
  real w;                   /* Hz frequency and Angular frequency */
  real t_temp;
  real f_t = 0.0;           /* function value                     */

  w = 2.0*pi*FREQ;  /* Angular frequency */

  SINE_K1 = -1; /* negative first  half of sine wave, for a pull */
  SINE_K2 = -7; /* positive second half of sine wave, for a push */

  t_temp = fmod(t,(1.0/FREQ));
  /* First  half of sine wave */
  if( t_temp>=0.0 && t_temp<=(0.5/FREQ) )
  {
   f_t = (SINE_K1/(SINE_K1+SINE_K2))*sin(w*t);
  }
  /* Second half of sine wave */
  else if( t_temp>(0.5/FREQ) && t_temp<(1.0/FREQ) )
  {
   f_t = (SINE_K2/(SINE_K1+SINE_K2))*sin(w*t);
  }

  return f_t;
}  /* power_funct subroutine end */
```

Subroutine: coord_xform

The subroutine coord_xform modifies two points that are given to it, pu and pv. This is done by first finding the different angles between the x-axis and: the point to be transformed, the transformation vector, and finally the coordinates of the tail of the transformation vector. Once done, all of the points and vectors are rotated by the angle that was found for the transformation vector. The transformed point has the transformed base point subtracted from it as it is now the new (0,0) point.

```
/*
************************************************************************
Subroutine:  coord_xform

Modified:    08/08/2005
Created:     03/29/2005
Creator:     Capt Timothy R. Klein

Description: Transforms coordinates and vectors to give 2 sets of
             points.
************************************************************************
*/
void coord_xform(real   px, real   py, /*Coordinate to be Transformed*/
                 real  *pu, real  *pv, /*Transformed coordinate      */
                 real  v1x, real  v1y, /*Vector for transformation   */
                 real  p1x, real  p1y )/*Base of Vector              */
{
  *pu =  v1x*(px-p1x) + v1y*(py-p1y);
  *pv = -v1y*(px-p1x) + v1x*(py-p1y);

  return;

}  /* coor_xform subroutine end */
```

Subroutine: four_point

The subroutine four_point receives 4 points in no particular order and finds which sets of 2 points are adjacent to each other by finding the longest length between all of the points. The points are then grouped into 2 sets of 3 points; each set shares 2 points that were adjacent to each other. These points are sent to the subroutine three-point for line-straddle checking and finally integration.

```
/*
***********************************************************************
Subroutine:  four_point

Modified:    08/08/2005
Created:     03/21/2005
Creator:     Capt Timothy R. Klein

Description: Takes in 4 points in no particular order and finds
             adjacent points.  The points are then grouped into 3's,
             sharing 2 opposite sets of points, and sent to the
             three_point subroutine for integration.
***********************************************************************
*/
real four_point( real u1,real v1,
                 real u2,real v2,
                 real u3,real v3,
                 real u4,real v4,
                 real weight)
{
  /* Declare Local Point Variables */
  real temp_weight = 0.0;

  /* Calculate the vector magnitudes for all 6 vectors from 4 points */
  real v12_mag = sqrt( pow((u2-u1),2) + pow((v2-v1),2) );
  real v13_mag = sqrt( pow((u3-u1),2) + pow((v3-v1),2) );
  real v14_mag = sqrt( pow((u4-u1),2) + pow((v4-v1),2) );
  real v23_mag = sqrt( pow((u3-u2),2) + pow((v3-v2),2) );
  real v24_mag = sqrt( pow((u4-u2),2) + pow((v4-v2),2) );
  real v34_mag = sqrt( pow((u4-u3),2) + pow((v4-v3),2) );
  /* Determine the maximum magnitude */
  real maximum = v12_mag;
  if( maximum < v13_mag ) maximum=v13_mag;
  if( maximum < v14_mag ) maximum=v14_mag;
  if( maximum < v23_mag ) maximum=v23_mag;
  if( maximum < v24_mag ) maximum=v24_mag;
  if( maximum < v34_mag ) maximum=v34_mag;

  /* Determine opposite points */
  if( maximum==v12_mag || maximum==v34_mag )
  {
   temp_weight += three_point( u1,v1, u3,v3, u4,v4, temp_weight );
   temp_weight += three_point( u2,v2, u3,v3, u4,v4, temp_weight );
  }
  else if( maximum==v13_mag || maximum==v24_mag )
  {
   temp_weight += three_point( u1,v1, u2,v2, u4,v4, temp_weight );
   temp_weight += three_point( u3,v3, u2,v2, u4,v4, temp_weight );
  }
  else if( maximum==v14_mag || maximum==v23_mag )
  {
   temp_weight += three_point( u1,v1, u2,v2, u3,v3, temp_weight );
   temp_weight += three_point( u4,v4, u2,v2, u3,v3, temp_weight );
  }

  return temp_weight;

}  /* four_point subroutine end */
```

Subroutine: three_point

The subroutine three_point receives 3 points in no particular order and checks if the area enclosed by the points straddles any boundary lines set by the user. If the points are all contained within a boundary region, then volume integration is completed and the weight is returned.

Otherwise, the line and boundary intercepts are found. A new set of 3 points is generated along with a set of 4 points. Each set should lie on opposite sides of the boundary. The subroutine is recursive until all boundaries have not been crossed by a set of points.

```
/*
**********************************************************************
Subroutine:   three_point

Modified:     08/08/2005
Created:      03/21/2005
Creator:      Capt Timothy R. Klein

Description: Takes in 3 points in no particular order and looks to see
             if the area enclosed by the points contains any boundaries
             defined in x_cross and y_cross.  If there is a boundary,
             then the area will be parsed into a 3 point and 4 point set
             to undergo evaluation again.  If there is no boundary, then
             the 3 points containing the area will be passed to a
             numerical integrator to return the weight or volume in that
             area.
**********************************************************************
*/
real three_point(real u1,real v1,
                 real u2,real v2,
                 real u3,real v3,
                 real weight)
{
  /* Declare Local Point Variables */
  int    check = 0;
  real a1u, a1v, a2u, a2v;  /* Points on line */
  real temp_weight = 0.0;

  int n;
```

```
for(n=1;n<6;++n)
{
 /* Check to see if points are on both sides of line */
 if( (    line_side(n,u1,v1)<0.0 &&
          line_side(n,u2,v2)<0.0 &&
          line_side(n,u3,v3)>0.0 ) ||
     (    line_side(n,u1,v1)<0.0 &&
          line_side(n,u2,v2)>0.0 &&
          line_side(n,u3,v3)<0.0 ) ||
     (    line_side(n,u1,v1)>0.0 &&
          line_side(n,u2,v2)<0.0 &&
          line_side(n,u3,v3)<0.0 ) ||

     (    line_side(n,u1,v1)>0.0 &&
          line_side(n,u2,v2)>0.0 &&
          line_side(n,u3,v3)<0.0 ) ||
     (    line_side(n,u1,v1)>0.0 &&
          line_side(n,u2,v2)<0.0 &&
          line_side(n,u3,v3)>0.0 ) ||
     (    line_side(n,u1,v1)<0.0 &&
          line_side(n,u2,v2)>0.0 &&
          line_side(n,u3,v3)>0.0 )   )
 {

   check = 1;  /* area being parsed, no need to run integration */

   /* p1 and p2 on same side of line */
   if( line_side(n,u1,v1)*line_side(n,u2,v2) > 0.0 )
   {
     /* Using p3 as the vertex,                        */
     /* find (u,v)-coordinates where lines intercept */
     line_intercept(n, u3,v3, u1,v1, &a1u,&a1v);
     line_intercept(n, u3,v3, u2,v2, &a2u,&a2v);

     temp_weight += four_point ( u1, v1,  u2, v2,
                                 a1u,a1v, a2u,a2v, temp_weight );
     temp_weight += three_point( u3, v3,
                                 a1u,a1v, a2u,a2v, temp_weight );
   }

   /* p1 and p3 on same side of line */
   else if( line_side(n,u1,v1)*line_side(n,u3,v3) > 0.0 )
   {
     /* Using p2 as the vertex,                        */
     /* find (u,v)-coordinates where lines intercept */
     line_intercept(n, u2,v2, u1,v1, &a1u,&a1v);
     line_intercept(n, u2,v2, u3,v3, &a2u,&a2v);


     temp_weight += four_point ( u1, v1,  u3, v3,
                                 a1u,a1v, a2u,a2v, temp_weight );
     temp_weight += three_point( u2, v2,
                                 a1u,a1v, a2u,a2v, temp_weight );
   }
```

```
      /* p2 and p3 on same side of line */
      else if( line_side(n,u2,v2)*line_side(n,u3,v3) > 0.0 )
      {
        /* Using p1 as the vertex,                       */
        /* find (u,v)-coordinates where lines intercept */
        line_intercept(n, u1,v1, u2,v2, &a1u,&a1v);
        line_intercept(n, u1,v1, u3,v3, &a2u,&a2v);


        temp_weight += four_point ( u2, v2,  u3, v3,
                                    a1u,a1v, a2u,a2v, temp_weight );
        temp_weight += three_point( u1, v1,
                                    a1u,a1v, a2u,a2v, temp_weight );
      }

    }  /* line_side if check */
  }  /* n for loop */

  /* Area was not split along any boundaries, */
  /* integrate area for volume weight         */
  if(check == 0)
  {
    /* Passed all checks on boundaries,    */
    /* can now integrate area WRT function */
    temp_weight += volume_integration(u1,v1, u2,v2, u3,v3, temp_weight);
  }

  return temp_weight;

}  /* three_point subroutine end */
```

Subroutine: line_side

The subroutine line_side receives a point as well as the line number to compare against. The line locations are defined in the subroutine line_offset. The subroutine line_side outputs the distance from the line in either x or y coordinates, depending upon the line direction. If the number to be returned is negative, then the point is to either the left or below the line. The opposite is true for a number that is returned positive.

```
/*
***********************************************************************
Subroutine:  line_side

Modified:    08/08/2005
Created:     03/28/2005
Creator:     Capt Timothy R. Klein

Description: Gives the side of the line the point (u,v) is on.
             (-) is the left side, (+) is the right side
***********************************************************************
*/
real line_side(int n, real u, real v)
{
  real offset;
  real side = 0.0;

  offset = line_offset(n);

  switch(n)
  {
  case 1:
  case 2:
  case 3:
  case 4:
    side = u - offset;
    break;
  case 5:
  case 6:
    side = v - offset;
    break;
  }

  return side;

}  /* line_side subroutine end */
```

Subroutine: line_intercept

The subroutine line_intercept receives a set of points as well as the line number to compare against. Because this function is set as void, there is no return value. Instead, the changes made by the function are done directly to the pointers au and av. The line locations are defined in the subroutine line_offset. The points au and av are the coordinates of where the intersection of the line formed by (u1,v1) and (u2,v2) and the line defined by n when passed to the subroutine line_offset.

```
/*
************************************************************************
Subroutine:  line_intercept

Modified:    08/08/2005
Created:     03/28/2005
Creator:     Capt Timothy R. Klein

Description: Gives the cross-product of a vector (vx,vy) with tail at
             point(px,py) and the resulting vector (x-px,y-py) when
             point (x,y) is the head and point (px,py) is the tail.  For
             simplicity, (px,py)=(0,0), thus making the resulting vector
             (x,y)
************************************************************************
*/
void line_intercept(int n,               /* line number            */
                    real  u1, real  v1,  /* point p1 = (u1,v1)      */
                    real  u2, real  v2,  /* point p2 = (u2,v2)      */
                    real *au, real *av)  /* intercept point (au,av) */
{
  real offset;
  real u1_new,u2_new, v1_new,v2_new;

  offset = line_offset(n);

  switch(n)
  {
  case 1:
  case 2:
  case 3:
  case 4:
    u1_new = u1 - offset;  /* location WRT line */
    u2_new = u2 - offset;  /* location WRT line */
```

```
      if( (u1_new <= 0.0 && u2_new >= 0.0) ||
          (u1_new >= 0.0 && u2_new <= 0.0) )
      {
        *au = 0.0 + offset;
        *av = v1 + (u1_new/(u1_new-u2_new))*(v2-v1);
      }
      else
      {
        *au = 0.0;
        *av = 0.0;
      }
      break;

    case 5:
    case 6:
      v1_new = v1 - offset;  /* location WRT line */
      v2_new = v2 - offset;  /* location WRT line */
      if( (v1_new <= 0.0 && v2_new >= 0.0) ||
          (v1_new >= 0.0 && v2_new <= 0.0) )
      {
        *av = 0.0 + offset;
        *au = u1 + (v1_new/(v1_new-v2_new))*(u2-u1);
      }
      else
      {
        *au = 0.0;
        *av = 0.0;
      }
      break;
    }

    return;

  }  /* line_intercept subroutine end */
```

Subroutine: line_offset

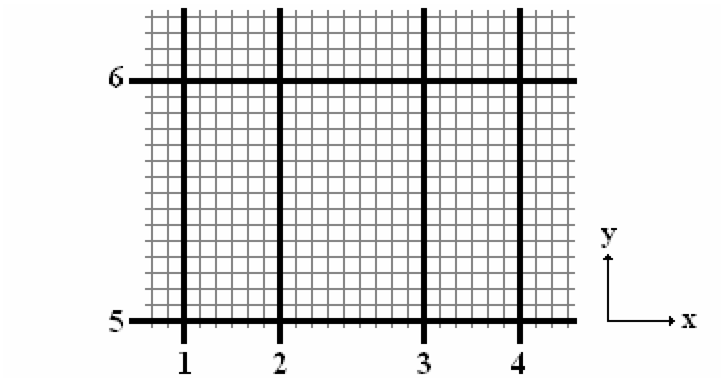


Figure 73
Line Number Reference Figure

The subroutine line_offset receives the line number and returns the value of the location of that line. This is an easy place to modify a globally used variable when making modifications. Parameterizations of experiments show that the DBD main wall jet effects are seen within 1 cm of the DBD device, with a glow surrounding the electrodes as well. Thus, the following bounding for the wall jet was assumed.

```
/*
***********************************************************************
Subroutine:  line_offset

Modified:    08/09/2005
Created:     03/28/2005
Creator:     Capt Timothy R. Klein

Description: Quick set area to line offsets.
***********************************************************************
*/
real line_offset(int n)
{
  real offset = 0.0;

  switch(n)
  {
```

```
case 1:  /* line1x */
  offset = -0.002; /* Original */
  switch(WScheme)
  {
  case 1:  /* Boueff&Pitchford */
    offset = -0.0001;  /*  -100 um */
    break;
  case 2:  /* Roy&Gaitonde     */
    offset = -0.0135;  /* -1.35 cm */
    break;
  default: /* Original         */
    offset = -0.0020;
    break;
  }
  break;

case 2:  /* line2x */
  offset =  0.000; /* Original */
  switch(WScheme)
  {
  case 1:  /* Boueff&Pitchford */
    offset =  0.0000;
    break;
  case 2:  /* Roy&Gaitonde     */
    offset = -0.0045;
    break;
  default: /* Original         */
    offset =  0.0000;
    break;
  }
  break;

case 3:  /* line3x */
  offset = +0.010; /* Original */
  switch(WScheme)
  {
  case 1:  /* Boueff&Pitchford */
    offset = +0.0006;   /* 600 um */
    break;
  case 2:  /* Roy&Gaitonde     */
    offset = +0.0040;   /* 0.4 cm */
    break;
  default: /* Original         */
    offset = +0.010;    /* 1.0 cm */
    break;
  }
  break;
```

```
    case 4:  /* line4x */
      offset = +0.015; /* Original */
      switch(WScheme)
      {
      case 1:  /* Boueff&Pitchford */
        offset = +0.0006;   /*  600 um */
        break;
      case 2:  /* Roy&Gaitonde     */
        offset = +0.0165;   /* 1.65 cm */
        break;
      default: /* Original         */
        offset = +0.010;    /* 1.00 cm */
        break;
      }
      break;

    case 5:  /* line1y */
      offset =  0.000; /* Original */
      switch(WScheme)
      {
      case 1:  /* Boueff&Pitchford */
        offset =  0.000;
        break;
      case 2:  /* Roy&Gaitonde     */
        offset =  0.000;
        break;
      default: /* Original         */
        offset =  0.000;
        break;
      }
      break;

    case 6:  /* line2y */
      offset = +0.003; /* Original */
      switch(WScheme)
      {
      case 1:  /* Boueff&Pitchford */
        offset = +0.00005;  /*  50 um */
        break;
      case 2:  /* Roy&Gaitonde     */
        offset = +0.005;    /* 0.5 cm */
        break;
      default: /* Original         */
        offset = +0.003;    /* 0.3 cm */
        break;
      }
      break;

    }

    return offset;

  }  /* line_offset subroutine end */
```

Subroutine: curve_y

The subroutine curve_y receives the x-coordinate for a point on a curve of the airfoil. The curve may be flat, as in the case of the flat-plate. However it may have curvature as described by this subroutine. The variable cord_length is used as a scaling factor for the location of the x-coordinate.

```
/*
***********************************************************************
Subroutine:  curve_y

Modified:    08/08/2005
Created:     03/21/2005
Creator:     Capt Timothy R. Klein

Description: Given a symmetric airfoil and a x-coordinate, function
             will return y.  cord_length is used as a scaling factor.
***********************************************************************
*/
real curve_y(real x_in, real cord_length)
{
  real y;

  real const D_1 = 0.0;
  real const D_2 = 0.0;
  real const D_3 = 0.0;
  real const D_4 = 9.0;

  if(F_OR_A == 0)  /* Flat-Plate */
  {
    real offset = 0.5*(cord_length-CORD) - Y_POSITION*CORD;
    y = 0.0 + offset;
  }
  else if(F_OR_A == 1)  /* Airfoil*/
  {
    real f  = D_1;  /* Maximum Camber                  */
    real xf = D_2;  /* Position of Maximum Camber      */
    real t  = 0.1*D_3 + 0.01*D_4;  /* % thickness/cord */

    /* return to normalized x-coordinates*/
    real x = x_in / cord_length;

    real c = 1.0;  /* Normalized Cord Length */

    real x1 = xf/c;
```

```
      real yc = c*
                  (f/c)*
                  pow(1.0/(1.0-x1),2)*
                  ( (1.0-2.0*x1) + 2.0*x1*(x/c) - (x*x)/(c*c) );

      real yt = c*5*t*( + ( 0.29690*pow(x,0.5) )
                        - ( 0.12600*pow(x,1   ) )
                        - ( 0.35160*pow(x,2   ) )
                        + ( 0.28430*pow(x,3   ) )
                        - ( 0.10150*pow(x,4   ) ) );

      /* y-coordinate with cord_length scaling factor */
      y = cord_length*(yc + yt);
    }

    return y;

  }  /* curve_y subroutine end */
```

Subroutine: curve_dy

The subroutine curve_dy receives the x-coordinate for a point on a curve of the airfoil. The curve may be flat, as in the case of the flat-plate. However it may have curvature as described by this subroutine. Either case will change the output given the x-coordinate, as this subroutine returns the slope at this point. The variable cord_length is used as a scaling factor for the location of the x-coordinate.

```
/*
***********************************************************************
Subroutine:  curve_dy

Modified:    04/05/2005
Created:     03/28/2005
Creator:     Capt Timothy R. Klein

Description: Given a symmetric airfoil and a x-coordinate, function
             will return slope=(dy/dx).  cord_length is used as a
             scaling factor.
***********************************************************************
*/
real curve_dy(real x_in, real cord_length)
{
  real dy_dx;

  real const D_1 = 0.0;
  real const D_2 = 0.0;
  real const D_3 = 0.0;
  real const D_4 = 9.0;

  if(F_OR_A == 0)  /* Flat-Plate */
  {
    dy_dx = 0.0;
  }
  else if(F_OR_A == 1)  /* Airfoil */
  {
    real f  = D_1;  /* Maximum Camber                */
    real xf = D_2;  /* Position of Maximum Camber    */
    real t  = 0.1*D_3 + 0.01*D_4;  /* % thickness/cord */

    /* return to normalized x-coordinates */
    real x = x_in / cord_length;
    real c = 1.0;  /* Normalized Cord Length */

    real x1 = xf/c;
    real dyc_dx = c*
                  (f/c)*
                  pow(1.0/(1.0-x1),2)*
                  ( (1.0-2.0*x1) + 2.0*x1*(1.0/c) - (2.0*x)/(c*c) );

    real dyt_dx = c*5*t*( + ( 0.5*0.29690*pow(x,-0.5) )
                          - (     0.12600             )
                          - ( 2.0*0.35160*pow(x, 1  ) )
                          + ( 3.0*0.28430*pow(x, 2  ) )
                          - ( 4.0*0.10150*pow(x, 3  ) ) );

    /* slope of line given x-coordinate           */
    /* normalized with cord_length scaling factor */
    dy_dx = (dyc_dx + dyt_dx);
  }

  return dy_dx;

} /* curve_dy subroutine end */
```

Subroutine: volume_integration

The subroutine volume_integration receives 3 points that do not cross over any boundaries and returns the weight via discrete integration.

```
/*
***********************************************************************
Subroutine:  volume_integration

Modified:    04/08/2005
Created:     03/22/2005
Creator:     Capt Timothy R. Klein

Description: Integrates the area under the curve for the area contained
             within the 3 given points.
***********************************************************************
*/
real volume_integration(real u1,real v1,
                        real u2,real v2,
                        real u3,real v3,
                        real weight)
{
  /* Declare Local Variables */
  real s,ds;
  real t,dt;
  real p1u,p1v, p2u,p2v, p3u,p3v, p4u,p4v;
  real temp_weight = 0.0;

  int i,j;

  /* Number of divisions in cell area = 0.5*num_div*(num_div+1) */
  int num_div = 50;

  /* Vector 1 */
  real v1u = u2-u1;
  real v1v = v2-v1;

  /* Vector 2 */
  real v2u = u3-u1;
  real v2v = v3-v1;

  real N = abs(v1u*v2v - v2u*v1v);

  s = 0.0;
  for(i=1;i<=num_div;++i)
  {

    ds = (1.0/(real)(num_div))*(1.0-0.0);
    t = 0.0;
    for(j=1;j<=(num_div-i+1);++j)
    {

      dt = (1.0/(real)(num_div-i+1))*(1.0-s);

      if( (t+dt)<(1.0-s) )  /* Square area's in the middle */
      {
        /* Coordinate Transformations */
        p1u = (v1u*(s   ) + v2u*(t   )) + u1;
        p1v = (v1v*(s   ) + v2v*(t   )) + v1;
        p2u = (v1u*(s+ds) + v2u*(t   )) + u1;
        p2v = (v1v*(s+ds) + v2v*(t   )) + v1;
        p3u = (v1u*(s   ) + v2u*(t+dt)) + u1;
        p3v = (v1v*(s   ) + v2v*(t+dt)) + v1;
        p4u = (v1u*(s+ds) + v2u*(t+dt)) + u1;
        p4v = (v1v*(s+ds) + v2v*(t+dt)) + v1;

        temp_weight += N*(1.0*ds*dt)*0.25*
                      ( weight_funct(p1u,p1v) +
                        weight_funct(p2u,p2v) +
                        weight_funct(p3u,p3v) +
                        weight_funct(p4u,p4v)  );

      }
      /* Triangle area's along the s=1-t edge */
      else if( (t+dt)>=(1.0-s) )
      {
        /* Coordinate Transformations */
        p1u = (v1u*(s   ) + v2u*(t   )) + u1;
        p1v = (v1v*(s   ) + v2v*(t   )) + v1;
        p2u = (v1u*(s+ds) + v2u*(t   )) + u1;
        p2v = (v1v*(s+ds) + v2v*(t   )) + v1;
        p3u = (v1u*(s   ) + v2u*(t+dt)) + u1;
        p3v = (v1v*(s   ) + v2v*(t+dt)) + v1;


        temp_weight += N*(0.5*ds*dt)*(1.0/3.0)*
                      ( weight_funct(p1u,p1v) +
                         weight_funct(p2u,p2v) +
                         weight_funct(p3u,p3v)  );

      }

      t += dt;
    }  /* j loop */

    s += ds;
  }  /* i loop */

  return temp_weight;

}  /* volume_integration subroutine end */
```

Subroutine: weight_funct

The subroutine weight_funct receives a point and returns its value. This subroutine was necessary for fast modification of the weighting function equation and how it acted on all sides of the boundaries.

```
/*
***********************************************************************
Subroutine:  weight_funct

Modified:    08/09/2005
Created:     03/22/2005
Creator:     Capt Timothy R. Klein

Description: 2D function of cell weight dependent on position.
***********************************************************************
*/
real weight_funct(real u, real v)
{
  /* Declare Local Variables */
  real f_xy;         /* function value        */
  real e,p,mid;

  real K_1, C_1U, C_1V, C_1;
  real K_2, C_2U, C_2V, C_2;
  real K_3, C_3U, C_3V, C_3;

  /*
  Switch for Boueff&Pitchford or Roy&Gaitonde Weighting-Scheme
    Boueff&Pitchford: WScheme = 1
    Roy&Gaitonde:     WScheme = 2
  */

  switch(WScheme)
  {
```

```
case 1:  /* Boueff&Pitchford */
  /* First  Function Weighting Constants */
  K_1  = + 10000.0;  /* Weight of Amplitude   */
  C_1U = + 50000.0;  /* Decent Rate constants */
  C_1V = + 80000.0;  /* Decent Rate constants */

  /* Second Function Weighting Constants */
  K_2  = + 10000.0;  /* Weight of Amplitude   */
  C_2U = +   100.0;  /* Decent Rate constants */
  C_2V = + 80000.0;  /* Decent Rate constants */

  /* -0.0001 <= u <   0.000   */
  if( u >= line_offset(1) && u < line_offset(2) )
  {
    /* -0.001  <= v <= +0.00005 */
    if( v <= line_offset(6) && v >= -0.001)
    {
      if(u<0.0) u=-u;
      if(v<0.0) v=-v;
      e    = 0.0 - u*C_1U - v*C_1V;
      f_xy = (exp(e))*K_1;
    }
    else
    {
      f_xy = 0.0;
    }
  }

  /*  0.000 <= u <= +0.00060 */
  else if( u >= line_offset(2) && u <= line_offset(3) )
  {
    /* -0.001 <= v <= +0.00005 */
    if( v <= line_offset(6) && v >= -0.001)
    {
      if(u<0.0) u=-u;
      if(v<0.0) v=-v;
      e    = 0.0 - u*C_2U - v*C_2V;
      f_xy = (exp(e))*K_2;
    }
    else
    {
      f_xy = 0.0;
    }
  }
  else  /* everywhere else f_xy=0.0 */
  {
    f_xy = 0.0;
  }
  break;
```

```
case 2:  /* Roy&Gaitonde */
  /* First  Function Weighting Constants */
  K_1  =   163.0 ;  /* Weight of Amplitude   */
  C_1  =     5.0 ;  /* Decent Rate constants */

  /* Second Function Weighting Constants */
  K_2  =  3200.0 ;  /* Weight of Amplitude   */
  C_2  =     2.75;  /* Decent Rate constants */

  /* Third  Function Weighting Constants */
  K_3  =   100.0 ;  /* Weight of Amplitude   */
  C_3U =     1.20*2;  /* Decent Rate constants */
  C_3V =     2.50*2;  /* Decent Rate constants */

  /* -0.0135 <= u <  -0.0045 */
  if( u >= line_offset(1) && u < line_offset(2) )
  {
    /* -0.0010 <= v <= +0.0050 */
    if( v <= line_offset(6) && v >= -0.001)
    {
      mid  = ( line_offset(1) + line_offset(2) ) / 2.0;
      u    = u - mid;
      p    = 0.0 - C_1*sqrt(u*u+v*v);
      if(u<0.0) u=-u;
      if(v<0.0) v=-v;
      p    = 0.0 - C_1*u - C_1*v;
      f_xy = K_1*(exp(p));
    }
    else
    {
      f_xy = 0.0;
    }
  }

  /* -0.0045 <= u <= +0.0040 */
  else if( u >= line_offset(2) && u <= line_offset(3) )
  {
    /* -0.0010 <= v <= +0.0050 */
    if( v <= line_offset(6) && v >= -0.001)
    {
      mid  = ( line_offset(2) + line_offset(3) ) / 2.0;
      u   -= mid;
      p    = 0.0 - C_2*sqrt(u*u+v*v);
      if(u<0.0) u=-u;
      if(v<0.0) v=-v;
      p    = 0.0 - u*100.0*C_2 - v*100.0*C_2;
      f_xy = K_2*pow(10.0,p);
    }
    else
    {
      f_xy = 0.0;
    }
  }
```

```
    /* +0.0040 <  u <= +0.0165 */
    else if( u >  line_offset(3) && u <= line_offset(4) )
    {
      /* -0.0010 <= v <= +0.0050 */
      if( v <= line_offset(6) && v >= -0.001)
      {
        mid  = ( line_offset(3) + line_offset(4) ) / 2.0;
        u   -= mid;
        if(u<0.0) u=-u;
        if(v<0.0) v=-v;
        p    = 0.0 - u*100.0*C_3U - v*100.0*C_3V;
        f_xy = 0.0 - K_3*pow(10.0,p);
      }
      else
      {
        f_xy = 0.0;
      }
    }
    else  /* everywhere else f_xy=0.0 */
    {
      f_xy = 0.0;
    }
    break;

  default:
    printf("Weighting-Scheme is neither Boueff&Pitchford or
            Roy&Gaitonde");

    /* First  Function Weighting Constants */
    K_1  =    1.0;  /* Weight of Amplitude   */
    C_1U =  800.0;  /* Decent Rate constants */
    C_1V =  800.0;  /* Decent Rate constants */

    /* Second Function Weighting Constants */
    K_2  =    7.0;  /* Weight of Amplitude   */
    C_2U =  100.0;  /* Decent Rate constants */
    C_2V =  800.0;  /* Decent Rate constants */

    /* -0.002 <= u <  0.000 */
    if( u >= line_offset(2) && u < line_offset(1) )
    {
      /* -0.001 <= v <= 0.010 */
      if( v <= line_offset(6) && v >= -0.001)
      {
        if(u<0.0) u=-u;
        if(v<0.0) v=-v;
        e    = 0.0 - u*C_1U - v*C_1V;
        f_xy = (exp(e))*K_1;
      }
      else
      {
        f_xy = 0.0;
      }
    }
    /*  0.000 <= u <= 0.010 */
    else if( u >= line_offset(1) && u <= line_offset(3) )
    {
      /* -0.001 <= v <= 0.010 */
      if( v <= line_offset(6) && v >= -0.001)
      {
        if(u<0.0) u=-u;
        if(v<0.0) v=-v;
        e    = 0.0 - u*C_2U - v*C_2V;
        f_xy = (exp(e))*K_2;
      }
      else
      {
        f_xy = 0.0;
      }
    }
    else  /* everywhere else f_xy=0.0 */
    {
      f_xy = 0.0;
    }
    break;
  }

  return f_xy;

}  /* weight_funct subroutine end */
```

VIII. Appendix C

The large number of data sets and resulting simulations to each set required an incredible amount of processing. To alleviate the monotony and presence of the operator in the computer lab, a PBS (Portable Batch System) script and a Journal script were created for each simulation to automate its setup and execution. This allowed for a quick initiation of the simulation from a remote site.

PBS Script

The operator first remotely logged-on to the AFIT Tahoe cluster computer and moved to the appropriate simulation set directory. A check by typing "qstat" was initiated to see the usage of the machine and to check on existing jobs already submitted to the queue. Submitting a job to the queue was done by typing "qsub" and the PBS script file name. This was then executed by the Tahoe cluster and assigned by the cluster to a node for processing. The following PBS script is the template example that was used for each simulation.

```
#!/bin/bash
#PBS -o out
#PBS -e error
#PBS -l nodes=1:ppn=2
#PBS -j oe

cd $PBS_O_WORKDIR

rm -R -f libudf_aoa000

fluent 2ddp -t2 -pnmpi -cnf=$PBS_NODEFILE -g -i
            journal_FLUENT_aoa000_152_tahoe.jou > FLUENT_aoa000_152.out
```

Some output is directed to the file "out", while any errors messages that occur are directed to the file "error". While only 1 node is used, 2 processors on that node are utilized. To utilize only 1 processor, simply change the switch "-t2" to "-t1" in the last line of the script. The directory is changed to that of where the PBS script is, and any previous UDF compilation library directory is removed. Fluent® is started in 2D with double precision variables. The *.jou journal file is fed to Fluent® to be executed and all screen output is directed to a *.out file. The PBS script will conclude once the journal file script has completed.

```
#!/bin/bash
#PBS -o out
#PBS -e error
#PBS -l nodes=1:ppn=2
#PBS -j oe

cd $PBS_O_WORKDIR

numbers='000
+02
+04
+06
+08
+10
+12
+14
+16'

prenamein  =journal_FLUENT_aoa
postnamein =_152_tahoe.jou
prenameout =FLUENT_aoa
postnameout=_152.out
libname    = libudf_aoa

for i in $numbers
do

echo $i UDF Library being Erased
rm -R -f $libname$i

echo $i Started
fluent 2ddp -t1 -pnmpi -cnf=$PBS_NODEFILE -g -i
            $prenamein$i$postnamein > $prenameout$i$postnameout

echo $i Finished
echo

done
```

For consecutive serial runs, the more efficient approach was performed. This allows for the simulations to be called one at a time for the entire Angle of Attack data set, which required tremendously less amounts of user interaction between runs and hence completes the set faster. However, the fastest approach is to run all of the set in parallel, assuming the processors and Fluent® software licenses are available.

Journal Script

The journal file is the heart of the simulation's running. The only input into this file is the information on the left of the below table. To help with understanding, the prompts that appear for some of the settings are shown to the right.

The settings for Fluent® reside in a folder system that can be accessed via text. To move up one folder level requires the command "q". Moving into a folder can be accomplished by typing its name while in the directory it exists in.

The main objective of the journal file is to setup the simulation, define the User Defined Functions (UDFs), execute the simulation, and write the appropriate data for later examination.

First we read in the Case and Data File.

`file`	cd file/ /file/
`rcd`	Read Case File & Data
`NACA0009_v07d6_base_FLUENT`	*.cas and *.dat Case File & Data respectively
`q`	cd .. /

The user defined memory is setup and the UDF library is compiled.

`define`	cd define/ /define/
`user-defined`	cd user-defined/ /define/user-defined/
`user-defined-memory`	Setting: user-defined-memory
`5`	Number of User-Defined Memory, UDM, locations
`compiled-functions`	Setting: compiled-functions
`compile`	load/unload/compile?
`libudf_aoa000`	Compiled UDF library name
`yes`	Continue?
`temp_mom_src_trm_FLUENT.c`	Give C-Source file names: First file name
`""`	Next file name
`""`	Give header file names: First file name
`compiled-functions`	Setting: compiled-functions
`load`	load/unload/compile
`libudf_aoa000`	UDF Library Name Angle of Attack set at 0 degrees
`q`	cd .. /define/

Species Transport is enabled to allow for the fluid Energy Equation to be used. This will allow for the input of the Temperature UDF Source, but is not required for the input of the Momentum UDF Source.

`models`	cd models/ /define/models/
`viscous`	cd viscous/ /define/models/viscous/
`spalart-allmaras?`	Setting: spalart-allmaras?
`yes`	Enable the Spalart-Allmaras Turbulence model?
`q`	cd .. /define/models/
`species`	cd species/ /define/models/species/
`species-transport?`	Setting: species-transport?
`yes`	Enable the species transport model?
`mixture-template`	Select an available mixture material. (mixture-template)
`q`	cd .. /define/models/
`q`	cd .. /define/

The boundary conditions for the velocity inlet are set.

`boundary-conditions`	cd /define/boundary-conditions/
`velocity-inlet`	Setting: velocity-inlet
`velocity-inlet-6`	zone id/name
`yes`	Velocity Specification Method: Magnitude and Direction?
`yes`	Reference Frame: Absolute
`no`	Use Profile for Velocity Magnitude?
`15.2`	Velocity Magnitude (m/s)

`no`	Use Profile for X-Component of Flow Direction?
`+1.000000000000`	X-Component of Flow Direction $\cos(a)$, α = Angle of Attack
`no`	Use Profile for Y-Component of Flow Direction?
`+0.000000000000`	Y-Component of Flow Direction $\sin(a)$, α = Angle of Attack
`no`	Use Profile for Temperature?
`288.15`	Temperature (k)
`yes`	Turbulence Specification Method: Modified Turbulent Viscosity
`no`	Use Profile for Modified Turbulent Viscosity?
`0.001`	Modified Turbulent Viscosity (m2/s)
`no`	Use Profile for h2o mass fraction?
`0`	h2o mass fraction
`no`	Use Profile for o2 mass fraction?
`0`	o2 mass fraction

The boundary conditions for the fluid are set. The UDF Source Terms for Energy, X & Y Momentum are tied in as well.

`fluid`	Setting: fluid
`fluid`	zone id/name
`yes`	Specify source terms?
`no`	Use Constant Mass (kg/m3-s) source?
`no`	Use UDF for Mass (kg/m3-s) source?
`no`	Use Constant X Momentum (n/m3) source?
`yes`	Use UDF X Momentum (n/m3) source?
`"x_momentum_source::libudf_aoa000"`	udf-name

`no`	Use Constant Y Momentum (n/m3) source?
`yes`	Use UDF Y Momentum (n/m3) source?
`"y_momentum_source::libudf_aoa000"`	udf-name
`no`	Use Constant Modified Turbulent Viscosity (kg/s2-m) source?
`no`	Use UDF Modified Turbulent Viscosity (kg/s2-m) source?
`no`	Use Constant h2o (kg/m3-s) source?
`no`	Use UDF h2o (kg/m3-s) source?
`no`	Use Constant o2 (kg/m3-s) source?
`no`	Use UDF o2 (kg/m3-s) source?
`no`	Use Constant Energy (w/m3) source?
`yes`	Use UDF Energy (w/m3) source?
`"temp_source::libudf_aoa000"`	udf-name
`no`	Specify fixed values?
`yes`	Motion Type: Stationary?
`0`	X-Origin of Rotation-Axis (m)
`0`	Y-Origin of Rotation-Axis (m)
`no`	Deactivated Thread
`no`	Laminar zone?
`no`	Porous zone?
`q`	cd .. /define/

Setting the material properties of the fluid, air.

`materials`	cd materials/ /define/materials/
`change-create`	Setting: change-create
`air`	material-name>

`air`	material name
`yes`	air is fluid change Density?
`constant`	Density methods: new method
`0.993`	value (kg/m3)
`no`	change Cp (Specific Heat)?
`no`	change Thermal Conductivity?
`yes`	change Viscosity?
`constant`	Viscosity methods: new method
`1.70e-5`	value (kg/m-s)
`no`	change Molecular Weight?
`no`	change L-J Characteristic Length?
`no`	change L-J Energy Parameter?
`no`	change Thermal Expansion Coefficient?
`no`	change Degrees of Freedom?
`no`	change Speed of Sound?
`q`	cd .. /define/

Setting the operating pressure of the simulation.

`operating-conditions`	cd operating-conditions/ /define/operating-conditions/
`operating-pressure`	Setting: operating-pressure
`78669`	operating pressure (pascal)
`q`	cd .. /define/
`q`	cd .. /

To increase the speed at which the simulations could be performed, parallelization on 2 processors was implemented. This snippet partitions the grid.

`parallel`	cd parallel/ /parallel/
`partition`	cd partition/ /parallel/partition/
`auto`	cd auto/ /parallel/partition/auto/
`use-case-file-method`	Setting: use-case-file-method
`yes`	use case-file partition method?
`q`	cd .. /parallel/partition/
`q`	cd .. /parallel/
`q`	cd .. /

To ensure accurate reporting results for the coefficients of lift and drag, reference values need to be set correctly.

`report`	cd report/ /report/
`reference-values`	cd reference-values/ /report/reference-values/
`compute`	cd compute/ /report/reference-values/compute/
`velocity-inlet`	Setting: velocity-inlet
`velocity-inlet-6`	zone id/name
`q`	cd .. /report/reference-values/
`area`	Setting: area
`0.202`	reference area (m2)
`q`	cd .. /report/
`q`	cd .. /

The correct time step needed to be set.

`solve`	cd solve/ /solve/
`set`	cd set/ /solve/set/
`time-step`	Setting: time-step
`0.001`	time step (s)
`q`	cd .. /solve/

Output of the coefficient of lift to a file is setup.

`monitors`	cd monitors/ /solve/monitors/
`force`	cd force/ /solve/monitors/force/
`lift-coefficient`	Setting: lift-coefficient
`yes`	monitor cl?
`4`	zone id/name(1)
`()`	zone id/name(2)
`no`	print cl data?
`yes`	write cl data?
`"cl-history_aoa000"`	cl data file name?
`no`	plot cl data?
`no`	plot per zone?
`+0.000000000000`	x-component of lift vector $\sin(a)$, α = Angle of Attack
`+1.000000000000`	y-component of lift vector $\cos(a)$, α = Angle of Attack

Output of the coefficient of drag to a file is setup.

`drag-coefficient`	Setting: drag-coefficient
`yes`	monitor cd?
`4`	zone id/name(1)
`()`	zone id/name(2)
`no`	print cd data?
`yes`	write cd data?
`"cd-history_aoa000"`	cd data file name?
`no`	plot cd data?
`no`	plot per zone?
`+1.000000000000`	x-component of lift vector $\cos(a)$, α = Angle of Attack
`+0.000000000000`	y-component of lift vector $\sin(a)$, α = Angle of Attack
`q`	cd .. /solve/monitors/
`q`	cd .. /solve/
`initialize`	cd initialize/ /solve/initialize/

Default reference values for pressure and velocity were setup and the flow initialized with these values.

`set-defaults`	cd set-defaults/ /solve/initialize/ set-defaults/
`pressure`	Setting: pressure
`0`	Default value for Gauge Pressure
`x-velocity`	Setting: x-velocity
`0`	Default value for X Velocity
`y-velocity`	Setting: y-velocity

`0`	Default value for Y Velocity
`q`	cd .. /solve/initialize/
`initialize-flow`	Setting: initialize-flow
`q`	cd .. /solve/
`q`	cd .. /

Now, just prior to starting the iterative time solving, the weighting function is called.

`define`	cd define/ /define/
`user-defined`	cd user-defined/ /define/user-defined/
`execute-on-demand`	Setting: execute-on-demand
`"cell_weight_on_demand::libudf_aoa000"`	Execute on demand function name
`q`	cd .. /define/
`q`	cd .. /

Start solver for 1500 time steps with a maximum of 20 iterations per time step.

`solve`	cd solve/ /solve/
`dual-time-iterate`	Setting: dual-time-iterate
`1000`	Number of physical time steps
`20`	Number of iterations per time step
`q`	cd .. /

Once solution has completed all of the time steps, write the Case and Data to a file for later review.

`file`	cd file/ /file/
`wcd`	Write Case & Data File
`NACA0009_v07d6_aoa000_FLUENT_end`	case/data file name
`yes`	OK to overwrite?
`q`	cd .. /

Exit the program.

`exit`	exit program

Bibliography

[1] Hilbun, William M., "Workshop on Modeling Dielectric Barrier Discharges for Plasma Actuators." PowerPoint presentation, 13 May 2004, slide 7.

[2] Enloe, C.L., McLaughlin, T.E., VanDyken, R.D., Kachner, K.D., Jumper, E.J., Corke, T.C. "Mechanisms and Responses of a Single Dielectric Barrier Plasma". Technical Paper 2003-1021 (AIAA Press, Reno, 2003).

[3] Hilbun, William M., "Computational Modeling of Plasma Actuators." PowerPoint presentation, 7 Nov. 2003, slides 7,9.

[4] Post, Martiqua L. and Thomas C. Corke, "Separation Control on High Angle of Attack Airfoil Using Plasma Actuators", Technical Paper 2003-1024 (AIAA Press, Washington DC, 2003).

[5] Roth, J. Reece, "Physics of Plasmas 10(5)", May 2003

[6] Van Dyke, Milton, "An Album of Fluid Motion", Copyright 1982, Nineth printing October 2003, The Parabolic Press, Stanford, California, Page 91.

[7] Corke, Thomas C.; Jumper, Eric J.; Post, Martiqua L.; Orlov, Dmitriy; McLaughlin, Thomas E.. "Application of Weakly-Ionized Plasmas as Wing Flow-Control Devices", Technical Paper 2002-0350 (AIAA Press, Washington DC, 2003).

[8] Font, Gabriel I.. "Boundary Layer Control with Atmospheric Plasma Discharges", Technical Paper 2004-3574 (40th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit 11-14 July 2004, Fort Lauderdale, FL).

[9] http://www.nd.edu/~che1/control.html

[10] Corke, Thomas C.; Orlov, Dmitriy M.. "Numerical Simulation of Aerodynamic Plasma Actuator Effects". Technical Paper 2005-1083 (AIAA Press, Reno, 2005).

[11] White, Viscous Fluid Flow 2nd Ed., McGraw-Hill 1991.

[12] http://www.aae.uiuc.edu/m-selig/ads/coord_database.html#N

[13] http://www-berkeley.ansys.com/cfd/naca.html#03

[14] O'Neill, Charles R.. "Determination of Flight Stability Coefficients Using a Finite Element CFD", Oklahoma State University, Stillwater, OK.

[15] Selig, M., University of Illinois at Urbana-Champaign. http://www.nasg.com/afdb/show-airfoil-e.phtml?id=886

[16] Boeuf, J.P., Pitchford, L.C., "Electrohydrodynamic Force and Aerodynamic Flow Acceleration in Surface Dielectric Barrier Discharge", Journal of Applied Physics 97 103307 (2005).

[17] Roy, S., Gaitonde, D.V., "Multidimensional Collisional Dielectric Barrier Discharge for Flow Separation Control at Atmosphereic pressures". Technical Paper 2005-4631 (AIAA Press, Toronto, Canada).

[18] Newcamp, J. “Effects of Boundary Layer Flow Control Using Plasma Actuator Discharges”. AFIT Thesis Defense, 25 August 2005.

[19] Font, G.I., Jung, S., Enloe, C.L., McLaughlin, T.E., Morgan, W.L., Baughn, J.W. “Simulation of the Effects of Force and Heat Produced by a Plasma Actuator on Neutral Flow Evolution”. Technical Paper 2006-167 (AIAA 2006-167).

[20] Likhanskii, A.V. Shneider, M.N., Macheret, S.O., Miles, R.B. “Modeling of Interaction Between Weakly Ionized Near-Surface Plasmas and Gas Flow”. Technical Paper 2006-1204 (AIAA 2006-1204).

Vita

Captain Klein graduated from Wayzata High School and entered undergraduate studies at Michigan Technological University where he graduated Cum Laude with a Bachelor of Science degree in Electrical Engineering. He was commissioned through the Detachment 400 AFROTC at Michigan Technological University where he was recognized with the Blue Chip Award.

His first assignment was at Patrick AFB as a program engineer and manager for the Eastern Launch Range's Launch Safety Systems. For his second assignment, he was assigned to the Air Force Research Laboratory's Sensor Directorate, Wright-Patterson AFB, where he served as a project technical lead. While stationed at Wright-Patterson AFB, he simultaneously worked at the Sensor Directorate and attended the Air Force Institute of Technology's Graduate School of Engineering and Management. Upon graduation, he will continue to work for the Sensor Directorate developing plasma antennas and maintaining the technical lead role he is currently employed in.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 02-16-2006	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From - To)* Sep 2003 - Mar 2006

4. TITLE AND SUBTITLE
Macroscopic Computational Model of
Dielectric Barrier Discharge Plasma Actuators

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Klein, Timothy R., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GAP/ENP/06-07

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Recent progress in the generation and sustainment of gas discharges at atmospheric pressure has energized research in the field of plasma-aerodynamics. Plasma actuators are promising devices that achieve flow control with no moving parts, do not alter the airfoil shape and place no parts in the flow. The operation of a plasma actuator is examined using a macroscopic (force and power addition) computational fluid dynamic model of a dielectric barrier discharge, DBD, in Fluent®. A parametric approach is adopted to survey the range of requisite magnitudes of momentum and energy delivered to the flow field and to identify the effects of this localized momentum and energy addition on the flow characteristics. Simulations consider the initiation and control of flow over a flat plate in a low velocity fluid. The simulation velocity profiles are compared with the experimental observations of Corke (AIAA 2002-0350) as well as simulations of Font (AIAA 2004-3574), Boeuf and Pitchford (JAP 97 103307 2005), and Roy and Gaitonde (AIAA 2005-4631). The simulation is extended from a flat plate simulation to examine the flow modification over an airfoil. Flow characteristics of lift and drag are compared with experimental results of Post and Corke (AIAA 2003-1024) and the compatible energy/momentum addition is identified. Energy and momentum values are then compared and related to characteristic values arising in DBD operation.

15. SUBJECT TERMS
Dielectric Barrier Discharge, Plasma Actuators

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. William F. Bailey, ENP
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U	UU	162	(937)255-3636 x4501 e-mail : William.Bailey@afit.edu