2013

THE DETERMINATION OF DRAG AND LIFT FORCES OF A RIGID BEAM LOCATED WITHIN A TURBULENT BOINDARY LAYER BY MEANS OF LARGE EDDY SIMULATION MODELING

Santiago Helman
CUNY City College

How does access to this work benefit you? Let us know!

Follow this and additional works at: http://academicworks.cuny.edu/cc_etds_theses

Part of the Mechanical Engineering Commons

Recommended Citation


This Thesis is brought to you for free and open access by the City College of New York at CUNY Academic Works. It has been accepted for inclusion in Master’s Theses by an authorized administrator of CUNY Academic Works. For more information, please contact AcademicWorks@cuny.edu.
THE DETERMINATION OF DRAG AND LIFT FORCES OF A RIGID BEAM LOCATED WITHIN A TURBULENT BOUNDARY LAYER BY MEANS OF LARGE EDDY SIMULATION MODELING

THESIS

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering (Mechanical)

at

The City College
of the
City University of New York

by

Santiago Helman
January 2013

Approved:

Professor Yiannis Andreopoulos, Thesis Advisor

Professor Feridun Delale, Chairman
Department of Mechanical Engineering
Table of contents

List of figures………………………………………………………………………………3
List of Tables………………………………………………………………………………5
Abstract…………………………………………………………………………………..6
Introduction………………………………………………………………………………6
Procedure and results
   Section I. Generation of custom boundary condition……………………………9
   Section II. Computation of drag and lift forces acting on a rigid beam………20
Discussion of results…………………………………………………………………..33
Conclusions……………………………………………………………………………34
Appendix……………………………………………………………………………….36
References……………………………………………………………………………….47
List of Figures

Figure 1. Proposed boundary layer domain.........................................................10
Figure 2. Domain boundary conditions..............................................................10
Figure 3. Wind tunnel set up utilized.................................................................11
Figure 4. R.m.s values of axial, normal and spanwise velocity components
normalized by the friction velocity ut..............................................................12
Figure 5. Simplified velocity fluctuation distribution..........................................13
Figure 6. Implemented boundary condition and generated flow field....................14
Figure 7a. Velocity field generated in an extended domain.................................14
Figure 7b. Pressure field generated in an extended domain...............................14
Figure 8a. X-dir velocity at x=3.5m..................................................................15
Figure 8b. Y-dir velocity at x=3.5m..................................................................16
Figure 8c. Z-dir velocity at x=3.5m..................................................................16
Figure 9. Recycling method for turbulence generation........................................17
Figure 10a. Velocity field using directMapped boundary condition at t=3.206s......17
Figure 10b. Pressure field using directMapped boundary condition at t=3.206s.....18
Figure 10c. Velocity field using directMapped boundary condition at t=4.508s......18
Figure 10d. Pressure field using directMapped boundary condition at t=4.508s......19
Figure 11a. Vorticity magnitude using directMapped boundary condition
at t=3.206s......................................................................................................20
Figure 11b. Vorticity magnitude using directMapped boundary condition
at t=4.508s......................................................................................................20
Figure 12a. Proposed geometry (Side view).........................................................21
Figure 12b. Proposed geometry (Plan view).........................................................21
Figure 13. Meshed beam and support bracket......................................................21
Figure 14a. Holder only, lift and drag. Uf=11m/s, δ=100mm, D=40mm..............23
Figure 14b. Beam only, lift and drag. Uf=11m/s, δ=100mm, D=40mm..............23
Figure 14c. Beam and holder, lift and drag. Uf=11m/s, δ=100mm, D=40mm......23
Figure 15a. Holder only, lift and drag. Uf=8m/s, δ=110mm, D=40mm..............24
Figure 15b. Beam only, lift and drag. Uf=8m/s, δ=110mm, D=40mm..............24
Figure 15c. Beam and holder, lift and drag. Uf=8m/s, δ=110mm, D=40mm......24
Figure 16a. Holder only, lift and drag. Uf=5m/s, δ=120mm, D=40mm..............25
Figure 16b. Beam only, lift and drag. Uf=5m/s, δ=120mm, D=40mm..............25
Figure 16c. Beam and holder, lift and drag. Uf=5m/s, δ=120mm, D=40mm......25
Figure 17a. Holder only, lift and drag. Uf=2m/s, δ=150mm, D=40mm..............26
Figure 17b. Beam only, lift and drag. Uf=2m/s, δ=150mm, D=40mm..............26
Figure 17c. Beam and holder, lift and drag. Uf=2m/s, δ=150mm, D=40mm......26
Figure 18a. Holder only, lift and drag. Uf=11m/s, δ=100mm, D=20mm..............27
Figure 18b. Beam only, lift and drag. Uf=11m/s, δ=100mm, D=20mm..............27
Figure 18c. Beam and holder, lift and drag. Uf=11m/s, δ=100mm, D=20mm......27
Figure 19a. Holder only, lift and drag. Uf=11m/s, δ=100mm, D=80mm..............28
Figure 19b. Beam only, lift and drag. Uf=11m/s, δ=100mm, D=80mm..............28
Figure 19c. Beam and holder, lift and drag. Uf=11m/s, δ=100mm, D=80mm......28
Figure 20a. Velocity field (m/s). Holder only. T=9.996s, Uf=11m/s,
δ=100mm, D=20mm.......................................................................................29
Figure 20b. Pressure field (Pa). Holder only. T=9.996s, Uf=11m/s,
δ=100mm, D=20mm.......................................................................................29
Figure 20c. Pressure field on holder (Pa). T=9.996s, Uf=11m/s, δ=100mm, D=20mm. ..........................30
Figure 21a. Velocity field (m/s). Beam only. T=9.996s, Uf=11m/s, δ=100mm, D=20mm. ..........................30
Figure 21b. Pressure field (Pa). Beam only. T=9.996s, Uf=11m/s, δ=100mm, D=20mm. ..........................30
Figure 21c. Pressure field on beam (Pa). T=9.996s, Uf=11m/s, δ=100mm, D=20mm. ..........................30
Figure 22a. Velocity field (m/s). Beam and holder. T=9.996s, Uf=11m/s, δ=100mm, D=20mm. ..........................31
Figure 22b. Pressure field (Pa). Beam and holder. T=9.996s, Uf=11m/s, δ=100mm, D=20mm. ..........................31
Figure 22c. Pressure field on beam and holder (Pa). T=9.996s, Uf=11m/s, δ=100mm, D=20mm. ..........................32
Figure 22a. Drag and lift summary for holder only. ..........................................................32
Figure 22b. Drag and lift summary for beam only. ..........................................................33
Figure 22c. Drag and lift summary for beam and holder. ..................................................33
List of tables.

Table 1. Case description ........................................................................................................22
Table 2. Case results ..................................................................................................................22
Abstract

Recent studies such as those conducted by Akaydin, Elvin and Andreopoulos in [1] have described and quantified the electrical power that can be extracted from a moving flow by the implementation of energy harvesters. In such study, a flexible cantilevered beam was fitted with a cylindrical prism at the free end and placed within a uniform flow field. Due to the vortex shedding generated by the prism (e.g. Von Karman vortices) the beam was found to oscillate about the undisturbed position. The beam in question was then outfitted with a piezoelectric material in order to translate the transient strain energy found within the beam to electrical energy.

Similarly, recent experiments performed in the wind tunnel at CCNY have suggested that energy harvesting can be performed by placing a flexible cantilevered beam within a turbulent boundary layer. The current paper intends to provide a basic understanding of the physics involved by computing the time dependent forces that such cantilevered beam experiences within a turbulent boundary layer.

In order to compute drag and lift forces, an inlet boundary condition capable of generating turbulence alongside LES (large eddy simulation) was developed and the flow characteristics that it generated compared with the recycling technique introduced by Lundt [2]. The modeling of the flow under investigation and the corresponding computation of forces were performed utilizing OpenFOAM open source CFD code.

Introduction

The implementation of computational fluid dynamics software has proven to be a very valuable tool in engineering. The utilization of commercial CFD packages allows for the modeling of complex flows that can interact with intricate geometries and in turn facilitate an understanding of the underlying physical phenomena. Their strength is their ability to implement a wide range of compressible and incompressible fluid solvers by means of sophisticated numerical discretization schemes. Although their implementation in industry appears to be undisputed based on their ease of use, in academia the utilization of commercial CFD packages is given great scrutiny due to the secrecy of their computer code. Although some commercial codes provide for the customization of some of their functionalities via user defined functions or UDF’s, the physical models and associated numerics adopted within the code cannot be accessed and modified. This constraint, in many cases, renders the software useless in the academic realm.

Fortunately, the development and successful growth of the open-source software community made possible the distribution of highly sophisticated programs that are free and modifiable at the source code level. One such program is OpenFOAM, a finite volume-based CFD package that includes a wide variety of compressible and incompressible solvers for both turbulent and laminar flows. Its modular nature combined with the ability to program in C++ language, makes it relatively simple for the user to customize specific portions of the code or create new functionality in order to fulfill the user’s need.

With regards to the modeling of turbulence, RANS (Raynolds averaged Navier-Stokes) modeling approach stands out as the premier choice amongst engineers. In this particular method the timed-averaged Navier-Stokes equations of motion are solved by modeling
the eddy viscosity term that arises as a result of the averaging process. As a result, the random fluctuations that are inherent in turbulent flows are averaged out; giving way to time-averaged quantities that can be computed and used in design. The opposite approach to turbulence can be implemented by the use of DNS or direct numerical simulation. In this approach the Navier-Stokes equations are solved in a very fine grid in order to resolve all the flow motions down to the Kolmogov’s scales. Although there is no term that needs to be modeled, the computational power required in DNS only allows for the modeling of very simple flows.

A middle ground to turbulence modeling can be found in LES or large eddy simulation. LES modeling acknowledges the observations summarized by the turbulent energy cascade described in [3], where the small dissipative eddies are seen as more universal than the large energy-carrying eddies. Based on this, a relatively coarse grid can be implemented in order to resolve large energy-carrying turbulent structures, while the more universal dissipative effect of small eddies is modeled. By performing a filtering operation, the velocity field is decomposed into the sum of a filtered or resolved component and a residual or subgrid-scale component.

The filtering operation is carried out by a filter kernel. The three most commonly used filters employed in LES are the following:

a. Sharp Fourier cutoff filter: 
\[ \tilde{G}(k) = \int_D G(x') e^{-ikx'} dx' = 1 \text{ if } k \leq \Pi / \Delta, 0 \text{ otherwise} \]

b. Gaussian filter: 
\[ \tilde{G}(x) = 6 \Pi \exp \left( \frac{-6x^2}{\Delta^2} \right) \]

c. Top hat filter: 
\[ \tilde{G}(k) = 1 / \Delta \text{ if } |x| \leq \Delta / 2, 0 \text{ otherwise} \]

where the required grid spacing \( h \) is proportional to the specified filter width \( \Delta \).

For the current study, a Gaussian filter was utilized.

The conservation equations for the filtered and sub-grid velocity fields are obtained by applying the filter operator to the Navier-Stokes equations for incompressible flow. The filters utilized in LES are spatially uniform; therefore the filter and differentiation operators commute. Considering this, the continuity equations are obtained as shown on [3]:

\[ \frac{\partial \tilde{u}_i}{\partial x_i} = \frac{\partial u_i}{\partial x_i} = 0 \text{ Filtered continuity equation} \]

Form the equation above, the following equation is obtained:

\[ \frac{\partial \tilde{u}_i}{\partial x_i} = \frac{\partial}{\partial x_i} (u_i - \bar{u}_i) = 0 \]
The above equation shows that the filtered velocity field $\bar{u}$ and the residual field $u'$ are both solenoidal, therefore:

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0 \text{ (filtered)} \quad (1) \quad \frac{\partial u'_i}{\partial x_i} = 0 \text{ (sub-grid)} \quad (2)$$

The momentum equation written in conservative form is derived by applying the filter to the incompressible flow momentum equation as shown below:

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial}{\partial x_j} \left[ (u_i u_j) \right] = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \nu \frac{\partial}{\partial x_j} \left[ \frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right] \quad (3)$$

where $\bar{p}$ is the filtered pressure field. The filtered term $\left( u_i u_j \right)$ is not equal to the product of the filtered velocities $\left( \bar{u}_i \bar{u}_j \right)$. The difference between $\left( u_i u_j \right)$ and $\left( \bar{u}_i \bar{u}_j \right)$ is defined as the residual-stress tensor:

$$\tau_{ij}^r = (u_i u_j) - (\bar{u}_i \bar{u}_j) \quad (4)$$

The residual kinetic energy is defined as $k_r = \frac{1}{2} \tau_{ij}^r$ and the anisotropic residual-stress tensor as $\tau_{ij}^r = \tau_{ij}^r - \frac{2}{3} k_r \delta_{ij}$

The isotropic residual stress is included in the modified filtered pressure $\bar{p} = p + \frac{2}{3} k_r$

Considering these definitions, the filtered momentum equation can be rewritten as shown on [3] as:

$$\overline{Du_j}{Dt} = \nu \frac{\partial^2 \bar{u}_j}{\partial x_i \partial x_i} - \frac{\partial \tau_{ij}^r}{\partial x_i} - \frac{1}{\rho} \frac{\partial \bar{p}}{\partial x_j} \quad (5)$$

where $\overline{Du_j}{Dt} = \frac{\partial}{\partial t} + \bar{u} \cdot \nabla$

As in the case of Reynolds-averaged Navier-Stokes (RANS), the equations for LES are unclosed. Closure is obtained by modeling the sub-grid stress tensor (SGS) $\tau_{ij}^r$. The residual stress tensor can be decomposed as proposed by Germano:

$$\tau_{ij}^r = \bar{L}_{ij}^0 + \bar{C}_{ij}^0 + \bar{R}_{ij}^0 \quad (6)$$

The terms shown on eq. (6) correspond to the following:
Leonard stresses: Represent the interaction between resolved scales that result in SGS contributions

\[ L^0_{ij} = \overline{u_i u_j} - \overline{u_i} \overline{u_j} \]  

Cross term: Represents interaction between resolved and unresolved scales

\[ C^0_{ij} = \overline{u_i' u'_j} + \overline{u'_i u_j} - \overline{u'_i} \overline{u'_j} - \overline{u_i'} \overline{u_j'} \]  

SGS Reynolds stresses: Represent interactions between small, unresolved scales

\[ R^0_{ij} = u'_i u'_j - u'_i' u'_j \]  

In this particular project, the Smagorinsky model was implemented and it’s summarized as follows [3]:

\[ \tau_{ij} = -\frac{1}{3} \tau_{kk} \delta_{ij} - 2(C_s \Delta)^2 |\bar{S}| S_{ij} \]  

The eddy viscosity is modeled as

\[ \mu_{sgs} = \rho(C_s \Delta)^2 |\bar{S}| \quad \bar{S} = \sqrt{2S_{ij} S_{ij}} \]  

The effective viscosity is equal to \( \mu_{eff} = \mu_{mol} + \mu_{sgs} \) and the Smagorinsky constant usually has the value of \( C_s = 0.1 - 0.2 \)

The Navier-Stokes equations for LES are solved numerically within OpenFOAM by employing a finite volume approach. As previously mentioned, the Smagorinsky model was utilized with standard \( C_s = 0.15 \).

**Procedure and Results**

Due to the nature of the problem under consideration, the procedures and results described within this section were subdivided into two distinct areas. Section I below describes the procedures and results involved in the generation of a customized boundary condition intended to generate turbulent boundary layers. Section II on the other hand describes the generation of the geometry used for the determination of the drag and lift forces that a rigid cantilevered beam and its corresponding support bracket experiences. The forces obtained in question were obtained and plotted for a wide range of flow regimes and beam locations.

### Section I. Generation of custom boundary condition

The inherent three-dimensional nature of turbulence and hence of LES, requires the generation of a 3D domain. Even though OpenFOAM provides its own meshing tool (SnappyHex mesh) its flexibility and ease of use proved to be a burden. For that reason, GAMBIT mesh generation software was utilized to create the computational mesh. One of the main advantages that GAMBIT offers is the ability to read external journal files (see Appendix A for journal file sample) that contain the necessary commands required
for mesh generation. This, in turn, proved to be a very quick and efficient tool when modifying similar mesh geometries.

The generated mesh was imported into OpenFOAM by using the fluent3DMeshToFoam command. The resulting mesh geometry was visualized using paraview by executing paraFoam in the command line. Figure 1 below shows a cross-section of the generated mesh along the centerline of the domain.

![Figure 1. Proposed boundary layer domain](image1)

The boundary conditions used for this simulation are shown schematically in Figure 2. The left and right boundaries of the domain were linked as “periodic” and the top boundary was set as “slip” where a zero gradient condition is implemented when the quantity is a scalar and, if the quantity being computed is a vector, the normal component will be set to zero and the tangential component will be such that the gradient is zero locally. For the bottom boundary, a wall boundary condition was imposed (no-slip velocity). The outlet was set as an inlet-outlet condition where the velocity field and pressure field was switched between a fixed value and zero-gradient depending on the direction of the velocity at the boundary [6].

![Figure 2. Domain boundary conditions](image2)
As previously stated, the implementation of a velocity inlet boundary condition capable of generating turbulent boundary layers is of paramount importance in the current study. In order to prescribe a user-defined boundary condition, an OpenFOAM utility called “groovyBC”[7] was utilized. GroovyBC constitutes a good example of the benefits that the open source community can provide as a result of collaboration between its users. Such utility allows users of OpenFOAM to implement user defined boundary conditions without the need to modify the program’s source code. Even though the code is free and open, modifications at the source code level require a deep understanding of the code and of C++ programming language. Additionally, the debugging of user defined source code proved to be very difficult due to the lack of useful debugging tools that can readily be associated to OpenFOAM.

Provided with groovyBC, the velocity boundary condition was implemented as to replicate the statistics and behavior of the velocity field observed in the turbulent boundary layers studied by Anant Honkan and Yiannis Andreopoulos in [8]. Their experiments were performed at CCNY wind tunnel facility schematically depicted below.

![Figure 3. Wind tunnel set up utilized by Honkan and Andreopoulos](image)

One of the main objectives of their paper was to perform time-dependent measurements of velocity, vorticity vector, rate-of-strain tensor and dissipation in turbulent boundary layers. As a byproduct of their thesis, measurements of fluctuations of velocity components were published. The data reported for the r.m.s values of axial, normal and spanwise velocity fluctuation components is shown below along with findings obtained by other researches.
Figure 4. R.m.s values of axial, normal and spanwise velocity components normalized by the friction velocity $u_t$. 
The velocity fluctuation information shown above was utilized to determine the intensity of fluctuations with respect to the bottom wall of the computational domain. As a representative example, the fluctuations for \( u' \) were studied. By observation of its corresponding graph, the following simplified fluctuation distribution curve was developed:

![Figure 5. Simplified velocity fluctuation distribution](image)

The peak of fluctuations of 0.32 m/s at 0.015m from the wall were determined using the frictional velocity \( u_t \) obtained from experiments in the wind tunnel for a free stream velocity of 3.2 m/s combined with the experimental information shown in Figure 4. The second peak of 0.24 m/s located at 0.03m from the wall was established as a result of multiple trial simulations. It’s important to mention that such peak was adopted in order to obtain results for the velocity field that more closely resembled those observed during actual PIV experiments performed in the wind tunnel [13].

The flow parameters utilized in the current study were the same as those used in [8] and are outlined below:

- \( U_e \) (Free stream velocity) = 3.2 m/s
- \( C_f \) (Drag coefficient) = 0.0033
- \( U_t \) (frictional velocity) = 0.13

Considering the fact that the fluctuation intensity within the boundary layer is normally distributed, a Gauss random number generator (generating numbers between -1 and +1) was multiplied by the velocity fluctuation distribution function depicted in Figure 5 at each grid point of the inlet plane. As a result, the random nature of turbulence and its intensity was reproduced numerically for each time step. The actual inlet velocity boundary condition in this project implemented (see Appendix) combined a mean velocity profile that obeyed the 1/7 power law [9] and the randomly generated fluctuations as described above. Figure 6 below shows the resultant velocity field imposed to the flow domain.
Figure 7a above shows the center plane of the velocity field generated by the utilization of the proposed boundary condition in a computational domain that is four times as long as the originally proposed. The pressure plot shown in Figure 7b corresponds to the velocity field portrayed in Figure 7a.

OpenFoam provides functionality as to specify points within the computational domain from which data can be extracted during run time. The following graphs (Figure 8)
portray velocity measurements at points located 3.5 meters from the inlet and at several vertical locations within the boundary layer.

![X velocity component](image.png)

**Figure 8a. X-dir velocity at x=3.5m**

It can be seen that there is a period of time in which the velocity changes from zero to a nominal value at which it oscillates. The reason for that derives from the fact that the velocity field was initially at rest for the entire flow field. The flow velocity takes up to 3 seconds to converge to its corresponding mean value. The x-direction velocity component reveals, as expected, that lower velocity magnitudes are found closer to the wall and that for points located at y=0.3m the mean velocity approaches the free stream velocity of 3.2m/s.

Similar observations can be made for the y-direction and z-direction components of velocity. The main difference that can be found when compared with the x-direction component of velocity is the fact that once the computation has converged, the components oscillate around zero mean. This can be explained by the fact that vertical and span-wise velocities do not have an effective mean velocity profile in addition to their fluctuations.
A more accepted method for generating turbulence in LES simulations is the one outlined by Lund [2]. In his method, a secondary computational domain is located upstream of the primary domain (e.g. the domain of interest). The secondary domain is utilized in order to generate a recycling and scaling of the velocity field. The set up of the two domains was carried out according to the diagram shown below. The “recycle” plane extracts the velocity information and feeds it back to the inlet. In addition, scaling of the velocity information is performed according to the law of the wall in the inner part and the defect law in the outer part of the boundary layer as described in [10].
Figure 9. Recycling method for turbulence generation

As a result, realistic boundary layer turbulence is generated with the added benefit of a lower cost in computational expense. This method can be implemented in OpenFOAM by using the provided “directMapped” boundary condition. This type of pre-programmed boundary condition follows the same procedure outlined by Lund. Simulations were performed in a domain of one meter in length and results for velocity and pressure are shown below for two different time steps.

Figure 10a. Velocity field using directMapped boundary condition at t=3.206s
Figure 10b. Pressure field using directMapped boundary condition at t=3.206s

Figure 10c. Velocity field using directMapped boundary condition at t=4.508s
As can be seen above, although somewhat similar, the velocity field obtained using directMapped does not exactly resemble the one obtained using the proposed method and shown on Figures 6 and 7. This could be attributed to the fact that no scaling of velocity or any other manipulation is performed by the proposed boundary condition other than those performed in an attempt to generate real life statistics for the flow. No consideration to continuity, skin friction or shear layer thickness is implemented resulting in a flow field that appears to be incorrect.

The pressure field on the other hand appears to be drastically different than the one obtained using the new proposed boundary condition and shown on Figure 7b. The pressure field portrayed in Figure 10b and 10d shows a very different picture where localized areas of low and high pressure appear to travel along the boundary layer. These pressure areas correspond to what seem to be vortex structures in the velocity field. In contrast, the pressure field shown in Figure 7b appears to vary gradually from inlet to outlet while its magnitude differs by 50 times. Below are corresponding plots of vorticity magnitude.
Section II. Computation of drag and lift forces acting on a rigid beam

The same tools and procedures described in Section I above were utilized in order to generate the required geometry. Three different journal files were generated (Beam only, support bracket only and combined beam & bracket) and implemented in order to determine the forces that each of the components are subjected to. The recycling technique developed by Lundt in [2] was used for the generation of a turbulent boundary layer. The computational domain was set to be 0.85m long, 0.25m wide and 0.5m high. The recycling plane location (see Figure 9) and the beam’s leading edge were located at 0.65m and 0.7m respectively downstream from the inlet boundary.
The proposed geometry is shown in Figure 12 above and its corresponding computational mesh obtained with Gambit was imported into OpenFOAM as described in Section I.

Several cases were set up and simulated in which the boundary layer thickness $\delta$, the location of the beam with respect to the bottom wall (distance “$D$”, see Figure 12) and the free stream velocity $U_f$ were modified. These cases were generated in order to monitor
the effects that the previously mentioned parameters have in the determination of drag and lift forces. The following charts describe the cases that were executed along with their corresponding results for the time average and standard deviation of drag & lift.

### Table 1. Case description

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
<th>Free stream velocity U (m/s)</th>
<th>Boundary layer thickness δ (mm)</th>
<th>Beam location w.r.t. bottom wall D (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Holder only</td>
<td>11</td>
<td>100</td>
<td>0.04</td>
</tr>
<tr>
<td>2</td>
<td>Beam only</td>
<td>11</td>
<td>100</td>
<td>0.04</td>
</tr>
<tr>
<td>3</td>
<td>Beam + Holder</td>
<td>11</td>
<td>100</td>
<td>0.04</td>
</tr>
<tr>
<td>4</td>
<td>Holder only</td>
<td>8</td>
<td>110</td>
<td>0.04</td>
</tr>
<tr>
<td>5</td>
<td>Beam only</td>
<td>8</td>
<td>110</td>
<td>0.04</td>
</tr>
<tr>
<td>6</td>
<td>Beam + Holder</td>
<td>8</td>
<td>110</td>
<td>0.04</td>
</tr>
<tr>
<td>7</td>
<td>Holder only</td>
<td>5</td>
<td>120</td>
<td>0.04</td>
</tr>
<tr>
<td>8</td>
<td>Beam only</td>
<td>5</td>
<td>120</td>
<td>0.04</td>
</tr>
<tr>
<td>9</td>
<td>Beam + Holder</td>
<td>5</td>
<td>120</td>
<td>0.04</td>
</tr>
<tr>
<td>10</td>
<td>Holder only</td>
<td>2</td>
<td>150</td>
<td>0.04</td>
</tr>
<tr>
<td>11</td>
<td>Beam only</td>
<td>2</td>
<td>150</td>
<td>0.04</td>
</tr>
<tr>
<td>12</td>
<td>Beam + Holder</td>
<td>2</td>
<td>150</td>
<td>0.04</td>
</tr>
<tr>
<td>13</td>
<td>Holder only</td>
<td>11</td>
<td>100</td>
<td>0.08</td>
</tr>
<tr>
<td>14</td>
<td>Beam only</td>
<td>11</td>
<td>100</td>
<td>0.08</td>
</tr>
<tr>
<td>15</td>
<td>Beam + Holder</td>
<td>11</td>
<td>100</td>
<td>0.08</td>
</tr>
<tr>
<td>16</td>
<td>Holder only</td>
<td>11</td>
<td>100</td>
<td>0.08</td>
</tr>
<tr>
<td>17</td>
<td>Beam only</td>
<td>11</td>
<td>100</td>
<td>0.08</td>
</tr>
<tr>
<td>18</td>
<td>Beam + Holder</td>
<td>11</td>
<td>100</td>
<td>0.08</td>
</tr>
</tbody>
</table>

### Table 2. Case results

<table>
<thead>
<tr>
<th>Case</th>
<th>Average drag (N)</th>
<th>Average lift (N)</th>
<th>Drag STD (N)</th>
<th>Lift STD (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.006968</td>
<td>0.005670</td>
<td>0.001264</td>
<td>0.0033</td>
</tr>
<tr>
<td>2</td>
<td>0.008215</td>
<td>0.005810</td>
<td>0.000360</td>
<td>0.004280</td>
</tr>
<tr>
<td>3</td>
<td>0.009217</td>
<td>0.002412</td>
<td>0.001406</td>
<td>0.004217</td>
</tr>
<tr>
<td>4</td>
<td>0.003496</td>
<td>0.000894</td>
<td>0.000069</td>
<td>0.000165</td>
</tr>
<tr>
<td>5</td>
<td>0.001219</td>
<td>-0.00036</td>
<td>0.000197</td>
<td>0.002284</td>
</tr>
<tr>
<td>6</td>
<td>0.004576</td>
<td>0.00216</td>
<td>0.000899</td>
<td>0.002812</td>
</tr>
<tr>
<td>7</td>
<td>0.001387</td>
<td>0.000378</td>
<td>0.000393</td>
<td>9.72E-05</td>
</tr>
<tr>
<td>8</td>
<td>0.000431</td>
<td>-0.00014</td>
<td>7.04E-05</td>
<td>0.001229</td>
</tr>
<tr>
<td>9</td>
<td>0.001817</td>
<td>0.000548</td>
<td>0.000364</td>
<td>0.000971</td>
</tr>
<tr>
<td>10</td>
<td>6.08E-05</td>
<td>2.18E-05</td>
<td>1.70E-05</td>
<td>5.59E-06</td>
</tr>
<tr>
<td>11</td>
<td>4.79E-05</td>
<td>-7.30E-05</td>
<td>1.10E-05</td>
<td>0.00131</td>
</tr>
<tr>
<td>12</td>
<td>0.000195</td>
<td>4.08E-05</td>
<td>4.35E-05</td>
<td>0.000169</td>
</tr>
<tr>
<td>13</td>
<td>0.002563</td>
<td>0.001220</td>
<td>0.000512</td>
<td>0.000220</td>
</tr>
<tr>
<td>14</td>
<td>0.001568</td>
<td>-0.000101</td>
<td>0.000265</td>
<td>0.002965</td>
</tr>
<tr>
<td>15</td>
<td>0.003655</td>
<td>0.002354</td>
<td>0.000648</td>
<td>0.002741</td>
</tr>
<tr>
<td>16</td>
<td>0.017450</td>
<td>0.001883</td>
<td>0.002522</td>
<td>0.000321</td>
</tr>
<tr>
<td>17</td>
<td>0.002879</td>
<td>-0.001635</td>
<td>0.000340</td>
<td>0.005720</td>
</tr>
<tr>
<td>18</td>
<td>0.019441</td>
<td>0.001717</td>
<td>0.002295</td>
<td>0.005250</td>
</tr>
</tbody>
</table>

The simulations described above were run in such a way that a total of 10 seconds worth of data was obtained. The data in question was acquired in 0.0085 second intervals and shown graphically on the figures below. The figures in question display a “time of convergence” from where the simulations begins at time $t=0s$ to a time $t=\Delta t$ at which the simulation results appear to stabilized. Based on this observation, $\Delta t$ was set equal to 4 seconds, therefore the average drag and lift results shown on Table 2 are computed using the interval of time $4s-10s$. 

22
Figure 14a. Holder only, lift and drag. Uf=11m/s, δ=100mm, D=40mm

Figure 14b. Beam only, lift and drag. Uf=11m/s, δ=100mm, D=40mm

Figure 14c. Beam and holder, lift and drag. Uf=11m/s, δ=100mm, D=40mm
Figure 15a. Holder only, lift and drag. Uf=8m/s, δ=110mm, D=40mm

Figure 15b. Beam only, lift and drag. Uf=8m/s, δ=110mm, D=40mm

Figure 15c. Beam and holder, lift and drag. Uf=8m/s, δ=110mm, D=40mm
Figure 16a. Holder only, lift and drag. $U_f=5\text{m/s}$, $\delta=120\text{mm}$, $D=40\text{mm}$

Figure 16b. Beam only, lift and drag. $U_f=5\text{m/s}$, $\delta=120\text{mm}$, $D=40\text{mm}$

Figure 16c. Beam and holder, lift and drag. $U_f=5\text{m/s}$, $\delta=120\text{mm}$, $D=40\text{mm}$
Figure 17a. Holder only, lift and drag. Uf=2m/s, δ=150mm, D=40mm

Figure 17b. Beam only, lift and drag. Uf=2m/s, δ=150mm, D=40mm

Figure 17c. Beam and holder, lift and drag. Uf=2m/s, δ=150mm, D=40mm
Figure 18a. Holder only, lift and drag. $U_f=11\text{m/s}$, $\delta=100\text{mm}$, $D=20\text{mm}$

Figure 18b. Beam only, lift and drag. $U_f=11\text{m/s}$, $\delta=100\text{mm}$, $D=20\text{mm}$

Figure 18c. Beam and holder, lift and drag. $U_f=11\text{m/s}$, $\delta=100\text{mm}$, $D=20\text{mm}$
Figure 19a. Holder only, lift and drag. $U_f=11 \text{m/s}$, $\delta=100 \text{mm}$, $D=80 \text{mm}$

Figure 19b. Beam only, lift and drag. $U_f=11 \text{m/s}$, $\delta=100 \text{mm}$, $D=80 \text{mm}$

Figure 19c. Beam and holder, lift and drag. $U_f=11 \text{m/s}$, $\delta=100 \text{mm}$, $D=80 \text{mm}$
The turbulent pressure and velocity fields corresponding to particular time steps can be seen below. The pressure distribution acting on the beam/holder assembly was also plotted. Figures 20a, 20b and 20c correspond to simulations performed for a beam holder only. In a similar fashion, Figures 21a, 21b and 21c correspond to simulations performed for a beam without a holder and Figures 22a, 22b and 22c correspond to simulations performed for the complete beam-holder assembly.

Figure 20a. Velocity field (m/s). Holder only. T=9.996s, Uf=11m/s, δ=100mm, D=20mm

Figure 20b. Pressure field (Pa). Holder only. T=9.996s, Uf=11m/s, δ=100mm, D=20mm
Figure 20c. Pressure field on holder (Pa). T=9.996s, Uf=11m/s, δ=100mm, D=20mm

Figure 21a. Velocity field (m/s). Beam only. T=9.996s, Uf=11m/s, δ=100mm, D=20mm

Figure 21b. Pressure field (Pa). Beam only. T=9.996s, Uf=11m/s, δ=100mm, D=20mm
Figure 21c. Pressure field on beam (Pa). $T=9.996\,\text{s}$, $U_f=11\,\text{m/s}$, $\delta=100\,\text{mm}$, $D=20\,\text{mm}$

Figure 22a. Velocity field (m/s). Beam and holder. $T=9.996\,\text{s}$, $U_f=11\,\text{m/s}$, $\delta=100\,\text{mm}$, $D=20\,\text{mm}$

Figure 22b. Pressure field (Pa). Beam and holder. $T=9.996\,\text{s}$, $U_f=11\,\text{m/s}$, $\delta=100\,\text{mm}$, $D=20\,\text{mm}$
The following graphs represent a summary of the results portrayed on Table 2. They aid in the visualization of the data obtained for each of the cases computed as they group results by case type. As an example, Figure 22a below depicts all the results obtained for the holder placed within the turbulent boundary layer as the different parameters are modified according to Table 1.

Figure 22a. Drag and lift summary for holder only.
**Discussion of results**

Considering the results obtained with the proposed boundary condition using groovyBC and the ones using the method outlined by Lund [2] and implemented in OpenFOAM using directMapped, it becomes evident that the proposed boundary condition does not generate correct results. The main indicator of such a statement resides in the comparison between both pressure fields. The pressure field obtained using the recycling technique shows the areas of high and low pressure within the boundary layer that are expected for this kind of flow. Also, the magnitude of pressure depicted in Figures 10b and 10d are in the order of magnitude that would be expected for this kind of flow regime. In contrast, the pressure field shown on Figures 10a and 10c corresponding to the proposed boundary condition shows a gradual pressure change throughout the domain with pressure values that are not in agreement with the physics at hand. Furthermore, a review of existing inflow generation techniques portrayed in [10] shows that attempts to generate boundary conditions that operate in a similar fashion to the one proposed in this paper fail to
replicate all the physics involved, producing pseudo- turbulence that is incomplete or inaccurate.

In contrast, the determination of drag and lift forces for the beam, holder and beam-holder assembly, by means of the turbulent recycling technique produced encouraging results. Experiments performed at CCNY’s wind tunnel using free stream velocities of the order of $U_f=11\text{ m/s}$ and utilizing similar geometries to the one implemented (see Figure 12) suggest that drag and lift forces described in this paper are in agreement with the physics at hand.

Examination of Figures 14 through 19 reveal the following:

1. When the holder is placed within the boundary layer, drag and lift forces are always positive
2. When only the beam is placed within the boundary layer, the drag force is as expected always positive; however the lift force appears to fluctuate about zero-lift. Furthermore, Figure 22b reveals that excluding Case#2, the time average of the lift is always negative (negative lift) suggesting that the net force acting on a beam placed within a turbulent boundary layer is always downwards.
3. Figures 17a, 17b and 17c depict inconsistent fluctuations in drag and lift as well as a larger convergence time. These figures suggest that as the free stream velocity decreases, the mesh resolution must be increased in order to properly resolve the boundary layers that arise around the beam in order to accurately determine the pressure field acting on the beam. This in turn, will allow for a more realistic computation of drag and lift forces at lower flow velocities

As was pointed out above in #2, the time average of the lift force appears to be negative. This interesting phenomenon deserves a special note since to this writer’s knowledge this observation has never been mentioned in literature. As suggested by professor Andreopoulos, this may be attributed to clockwise vortices impinging the beam’s leading edge resulting in a force that on average is oriented downwards.

The pressure field shown on Figures 21c and 22c show areas of increased pressure toward the trailing edge of the beam at around the clap support. Also, the pressure field throughout the beam appears to vary smoothly without scattered areas of low and high pressure. This appears to be contrary to what would be expected considering the random nature of the flow in question. Considering this, further studies should be carried out where the mesh density at the beam surface is increased in order to capture possible areas of low and high pressures that may not be resolved with the coarser mesh utilized in the current study. It’s important to mention that a drastic increase in mesh density would impose a computational burden since the code utilized is not programmed for parallel computing.

**Conclusions**

The proposed boundary condition did not perform as expected and therefore its application in the generation of turbulent boundary layers would provide inaccurate flow
characteristics. This is in accordance with relevant literature were similar approaches in the generation of turbulence were proposed but seldom proved to be successful. In the current study, even though capable of reproducing some of the physical aspects present in turbulent boundary layers, the boundary condition was proved to be not an acceptable method for turbulent inflow generation in LES. Considering the previous statements, the widely accepted method introduced by Lundt appears to be the premier choice for turbulent boundary layer generation in the framework of LES.

In light of the results obtained for a beam placed within the turbulent boundary layer, it appears that energy harvesting from the kind of flow regimes under consideration is feasible. The fluctuating nature of the lift force observed for the rigid beam under investigation suggest that the utilization of a deformable beam outfitted with piezoelectric material could translate deformations generated by the flow into useful electrical energy. The data extracted also suggest that a beam immerse into this kind of flow would experience high frequency random oscillations. The challenge in the extraction of electrical energy from such device appears to be found in how to control the oscillations in order to maximize power output.

Attempts were made as to implement a fluid-structure interaction model in the framework of OpenFOAM in order to investigate the behavior of deformable beams and their interaction with the flow. Due to bugs found within the code, the implementation of the pre-programmed turbulent recycling technique in combination with movable meshes was not possible. A great deal of effort was invested as to bypass the previously mentioned bug in the code without success. It became apparent that the harnessing of fluid-structure interaction simulations using turbulent modeling such as LES requires a high level of understanding of the OpenFOAM source code and C++ programming language. OpenFOAM documentation was found to be mediocre and web based OF forums, although useful, proved to be a very ineffective tool.
Appendix

1. GroovyBC script

The code below was implemented in the current study and was specified as a velocity boundary condition at the inlet. The limitations associated with groovyBC required the code to be written in a single line making the code hard to read. The reader is referred to [6] for a more in-depth understanding of the symbols used in programming using groovyBC.

```
FoamFile
{
    version 2.0;
    format ascii;
    class volVectorField;
    object U;
}

dimensions [0 1 -1 0 0 0 0];

internalField uniform (0 0 0);

boundaryField
{
    inlet
    {
        type groovyBC;
        variables "Ubl=3.2;h=.15;Ux=(pos().y <= .15 ? (Ubl*pow((pos().y/h),.1428)) : 3.2 ? (pos().y <= .01 ? 40*pos().y : (pos().y >.01 && pos().y <=.015 ? -16*pos().y+0.56 : -1.729*pos().y+0.3458)) : 0);Utotalx=Ux+(randNormal()*2)*ux;Uy=(1/100)*Ux;uy=(1/2.5)*ux;Utotaly=Uy+(randNormal()*2)*uy;uz=(1/2)*ux;Utotalz=(randNormal()*2)*uz;";
        valueExpression "vector (Utotalx, Utotaly, Utotalz)";
    }

    outlet
    {
        type inletOutlet;
        inletValue uniform (0 0 0);
        value uniform (0 0 0);
    }

    upperWall
    {
        type slip;
        //type fixedValue;
        //value uniform (3.2 0 0);
    }

    lowerWall
    {
```
{ 
  type fixedValue;
  value uniform (0 0 0);
}
left
{
  type ggi;
}
right
{
  type ggi;
}

2. Sample Gambit journal file utilized in the generation of the meshed geometries

$plateThickness = .0008
$plateLength = .1
$plateLocation = .08
$leadingEdgeLocation = 0.7
$plateLenght = .1
$plateWidth = .03
$clampThickness = 0.005
$clampLength = 0.01
$standSize = .005
$standLength = .03

vertex create coordinates 0 0 0
vertex create coordinates 0 ($plateLocation+$plateThickness/2+$clampThickness/2) 0
vertex create coordinates 0 ($plateLocation+$plateThickness/2) 0
vertex create coordinates 0 ($plateLocation+$plateThickness/2+$clampThickness/2) 0
vertex create coordinates 0.5 0
vertex create coordinates $leadingEdgeLocation 0 0
vertex create coordinates $leadingEdgeLocation ($plateLocation+$plateThickness/2+$clampThickness/2) 0
vertex create coordinates $leadingEdgeLocation ($plateLocation+$plateThickness/2) 0
vertex create coordinates $leadingEdgeLocation ($plateLocation+$plateThickness/2+$clampThickness/2) 0
vertex create coordinates $leadingEdgeLocation ($plateLocation+$plateThickness/2+$clampThickness/2) 0
vertex create coordinates $leadingEdgeLocation 0.5 0

/front of clamp
vertex create coordinates ($leadingEdgeLocation+$plateLength-$clampLength) 0 0
vertex create coordinates ($leadingEdgeLocation+$plateLength-$clampLength) ($plateLocation-$plateThickness/2-$clampThickness/2) 0
vertex create coordinates($leadingEdgeLocation+$plateLength$clampLength)
(SplateLocation-(SplateThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength$clampLength)
(SplateLocation+(SplateThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength$clampLength)
(SplateLocation+(SplateThickness/2)+(SclampThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength$clampLength) 0.5 0

/back of plate and clamp
vertex create coordinates($leadingEdgeLocation+$plateLength) 0 0
vertex create coordinates($leadingEdgeLocation+$plateLength) (SplateLocation-(SplateThickness/2)-(SclampThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength) (SplateLocation-(SplateThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength) (SplateLocation+(SplateThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength) (SplateLocation+(SplateThickness/2)+(SclampThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength) 0.5 0

/at front of stand
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength$standSize) 0 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength$standSize) (SplateLocation-(SplateThickness/2)-(SclampThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength$standSize) (SplateLocation-(SplateThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength$standSize) (SplateLocation+(SplateThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength$standSize) (SplateLocation+(SplateThickness/2)+(SclampThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength$standSize) 0.5 0

/at end of stand
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength) 0 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength) (SplateLocation-(SplateThickness/2)-(SclampThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength) (SplateLocation-(SplateThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength) (SplateLocation+(SplateThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength) (SplateLocation+(SplateThickness/2)+(SclampThickness/2)) 0
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength+$standLength) 0.5 0

/end of domain
vertex create coordinates($leadingEdgeLocation+$plateLength+$standLength+.03) 0 0
vertex create coordinates ($leadingEdgeLocation+$plateLength+$standLength+.03) ($plateLocation-(+$plateThickness/2)-($clampThickness/2)) 0
vertex create coordinates ($leadingEdgeLocation+$plateLength+$standLength+.03) ($plateLocation-($plateThickness/2)) 0
vertex create coordinates ($leadingEdgeLocation+$plateLength+$standLength+.03) ($plateLocation+$($plateThickness/2)) 0
vertex create coordinates ($leadingEdgeLocation+$plateLength+$standLength+.03) ($plateLocation+$($plateThickness/2)+($clampThickness/2)) 0
vertex create coordinates ($leadingEdgeLocation+$plateLength+$standLength+.03) 0.5 0

edge create straight "vertex.1" "vertex.2" "vertex.3" "vertex.4" "vertex.5" \ "vertex.6"
edge create straight "vertex.7" "vertex.8" "vertex.9" "vertex.10" "vertex.11" \ "vertex.12"
edge create straight "vertex.13" "vertex.14" "vertex.15" "vertex.16" \ "vertex.17" "vertex.18"
edge create straight "vertex.19" "vertex.20" "vertex.21" "vertex.22" \ "vertex.23"
edge create straight "vertex.23" "vertex.24"
edge create straight "vertex.25" "vertex.26" "vertex.27" "vertex.28" \ "vertex.29" "vertex.30"
edge create straight "vertex.1" "vertex.7" "vertex.13" "vertex.19" \ "vertex.25"
edge create straight "vertex.2" "vertex.8" "vertex.14" "vertex.20" \ "vertex.26"
edge create straight "vertex.3" "vertex.9" "vertex.15" "vertex.21" \ "vertex.27"
edge create straight "vertex.4" "vertex.10" "vertex.16" "vertex.22" \ "vertex.28"
edge create straight "vertex.5" "vertex.11" "vertex.17" "vertex.23" \ "vertex.29"
edge create straight "vertex.6" "vertex.12" "vertex.18" "vertex.24" \ "vertex.30"
edge create straight "vertex.25" "vertex.31"
edge create straight "vertex.26" "vertex.32"
edge create straight "vertex.27" "vertex.33"
edge create straight "vertex.28" "vertex.34"
edge create straight "vertex.29" "vertex.35"
edge create straight "vertex.30" "vertex.36"
edge create straight "vertex.31" "vertex.32" "vertex.33" "vertex.34" \ "vertex.35" "vertex.36"
edge create straight "vertex.31" "vertex.37"
edge create straight "vertex.32" "vertex.38"
edge create straight "vertex.33" "vertex.39"
edge create straight "vertex.34" "vertex.40"
edge create straight "vertex.35" "vertex.41"
edge create straight "vertex.36" "vertex.42"
edge create straight "vertex.37" "vertex.38" "vertex.39" "vertex.40" \
face create wireframe "edge.26" "edge.6" "edge.30" "edge.1" real
face create wireframe "edge.30" "edge.7" "edge.34" "edge.2" real
face create wireframe "edge.34" "edge.8" "edge.38" "edge.3" real
face create wireframe "edge.38" "edge.9" "edge.42" "edge.4" real
face create wireframe "edge.42" "edge.10" "edge.46" "edge.5" real
face create wireframe "edge.27" "edge.11" "edge.31" "edge.6" real
face create wireframe "edge.31" "edge.12" "edge.35" "edge.7" real
face create wireframe "edge.35" "edge.13" "edge.39" "edge.8" real
face create wireframe "edge.39" "edge.14" "edge.43" "edge.9" real
face create wireframe "edge.43" "edge.15" "edge.47" "edge.10" real
face create wireframe "edge.28" "edge.16" "edge.32" "edge.11" real
face create wireframe "edge.32" "edge.17" "edge.36" "edge.12" real
face create wireframe "edge.36" "edge.18" "edge.40" "edge.13" real
face create wireframe "edge.40" "edge.19" "edge.44" "edge.14" real
face create wireframe "edge.44" "edge.20" "edge.48" "edge.15" real
face create wireframe "edge.29" "edge.21" "edge.33" "edge.16" real
face create wireframe "edge.33" "edge.22" "edge.37" "edge.17" real
face create wireframe "edge.37" "edge.23" "edge.41" "edge.18" real
face create wireframe "edge.41" "edge.24" "edge.45" "edge.19" real
face create wireframe "edge.45" "edge.25" "edge.49" "edge.20" real
face create wireframe "edge.50" "edge.56" "edge.51" "edge.21" real
face create wireframe "edge.51" "edge.57" "edge.52" "edge.22" real
face create wireframe "edge.52" "edge.58" "edge.53" "edge.23" real
face create wireframe "edge.53" "edge.59" "edge.54" "edge.24" real
face create wireframe "edge.54" "edge.60" "edge.55" "edge.25" real
face create wireframe "edge.61" "edge.67" "edge.62" "edge.56" real
face create wireframe "edge.62" "edge.68" "edge.63" "edge.57" real
face create wireframe "edge.63" "edge.69" "edge.64" "edge.58" real
face create wireframe "edge.64" "edge.70" "edge.65" "edge.59" real
face create wireframe "edge.65" "edge.71" "edge.66" "edge.60" real

/left
volume create translate "face.1" "face.2" "face.3" "face.4" "face.5" "face.6" \  "face.7" "face.8" "face.9" "face.10" "face.11" "face.12" "face.13" \  "face.14" "face.15" "face.16" "face.17" "face.18" "face.19" "face.20" \  "face.21" "face.22" "face.23" "face.24" "face.25" "face.26" "face.27" \  "face.28" "face.29" "face.30" vector 0 0 0.075

/plate
volume create translate "face.40" "face.35" "face.45" "face.50" "face.55" \  "face.80" "face.75" "face.70" "face.65" "face.60" "face.90" "face.85" \  "face.95" "face.100" "face.105" "face.115" "face.110" "face.120" "face.125" \  "face.130" "face.140" "face.135" "face.145" "face.150" "face.155" \  "face.165" "face.160" "face.170" "face.175" "face.180" vector 0 0 (($PlateWidth/2)-\($ClampThickness/2)))
group create "plate and clamp" volume "volume.68" "volume.98" "volume.38" "volume.101" "volume.103" "volume.104" "volume.73" "volume.72" "volume.74" "volume.41" "volume.43" "volume.44"
group modify "plate and clamp" color "yellow"

group create "holder" volume "volume.79" "volume.78" "volume.77" "volume.84" "volume.83" "volume.82" "volume.81"
group modify "holder" color "yellow"

/here this is plate plus holder

physics create "plateFluid" btype "WALL" face "face.70" "face.193" "face.211" "face.216" "face.220" "face.343" "face.364" "face.369" "face.370" "face.493" "face.509" "face.519" "face.520" "face.399" "face.90" "face.95" "face.100" "face.394" "face.213" "face.389" "face.218" "face.384" "face.223" "face.231" "face.234" "face.235" "face.244" "face.245" "face.249" "face.250" "face.363" "face.373" "face.368" "face.550" "face.549" "face.545" "face.544" "face.390" "face.508" "face.535" "face.518" "face.534" "face.531" "face.523" "face.400" "face.395" "face.260" "face.270" "face.275" "face.285" "face.290" "face.295" "face.300" "face.408" "face.409" "face.413" "face.414" "face.415" "face.418" "face.419" "face.420" "face.423" "face.424" "face.425" "face.433" "face.434" "face.435" "face.438" "face.439" "face.440" "face.443" "face.444" "face.445"
"face.481" "face.636" "face.56" "face.226" "face.356" "face.511" "face.656" "face.81" "face.236" "face.381" "face.536" "face.681" "face.106" "face.261" "face.406" "face.561" "face.706" "face.131" "face.286" "face.586" "face.751" "face.156" "face.311" "face.456" "face.611" "face.756"

physics create "upperWall" btype "WALL" face "face.54" "face.204" "face.354" "face.504" "face.654" "face.79" "face.209" "face.379" "face.529" "face.679" "face.104" "face.254" "face.404" "face.554" "face.704" "face.129" "face.279" "face.429" "face.579" "face.729" "face.154" "face.304" "face.454" "face.604" "face.734" "face.179" "face.329" "face.479" "face.629" "face.779"

physics create "inlet" btype "VELOCITY_INLET" face "face.32" "face.187" "face.332" "face.482" "face.637" "face.37" "face.182" "face.337" "face.487" "face.632" "face.42" "face.192" "face.342" "face.492" "face.642" "face.47" "face.197" "face.347" "face.497" "face.647" "face.52" "face.202" "face.352" "face.502" "face.652"

physics create "outlet" btype "PRESSURE_OUTLET" face "face.758" "face.458" "face.613" "face.158" "face.313" "face.163" "face.168" "face.173" "face.178" "face.308" "face.318" "face.323" "face.328" "face.463" "face.468" "face.473" "face.478" "face.608" "face.618" "face.623" "face.628" "face.763" "face.768" "face.773" "face.778"
References