Computational Fluid Dynamics
using the SU^2 Code

Brady Metherall 100516905

Monday April 4, 2016

List of Figures and Tables

1 Introduction

Before the invention of high speed computers, and GPU computing, fluid dynamics was a two discipline field – theoretical, and experimental. However, over the past few decades, a new discipline has emerged, computational fluid dynamics. Computational fluid dynamics has grown to an equal footing with the other two sub-fields, although, it will never be a replacement. Computational fluid dynamics is essentially a perfect wind tunnel; both give important information about the fluid flow around a shape such as an airplane, both allow parameters such as the Reynolds number, temperature, or flow speed to be changed, and both are an effective way to optimize the aerodynamics of an object. Where computational fluid dynamics pulls ahead of wind tunnels is its cost, speed, ease of use, and range of operation. Changing the Reynolds number, or temperature in a computational fluid dynamics experiment is as easy as typing in a new number; it's not quite this simple in a wind tunnel. Furthermore, computational fluid dynamics has the ability to provide a broader range of data, as well as simulate impractical situations to test in a wind tunnel, such as the extreme speeds and temperatures a sub-orbital spacecraft would experience. Computational fluid dynamics experiments can even be done from anywhere in the world, by anyone, via remote access.

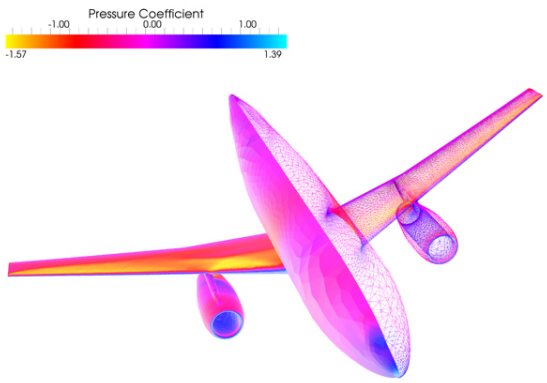


Figure 1: Computational fluid dynamics has a wide range of applications – passenger aircraft for instance [8].

Computational fluid dynamics has a wide variety of applications even today. There are some things that only computational fluid dynamics simulations can predict, such as how molten iron will flow into a mold in a manufacturing plant. Computational fluid dynamics plays an important role in other areas as well, including improving fuel efficiency of automobiles, optimizing pumps and turbines in dams, and even testing naval submarines and torpedoes. Computational fluid dynamics rivals other areas of physics and engineering as a vital area of research.

In this project we run basic computational fluid dynamics simulations for two dimensional flows. The Stanford University Unstructured (SU^2) code was used to run both laminar and turbulent flow simulations. We first examine some background theory, and numerical simplifications; our first experiment looks at a classic airfoil. Then, a static cylinder using both Euler's equations as well as the Navier-Stokes equations was simulated. Finally, we experimented with turbulent flow past a static cylinder and observed vortex shedding and a von Kármán vortex street.

2 Theory

2.1 Laminar Flow Equations

Non-viscous, ideal fluids obey two equations of motion collectively called Euler's equations. The first is the incompressibility condition

$$\nabla \cdot \mathbf{u} = 0, \tag{1}$$

which arises by assuming the volume of a fluid element, as well as its density are constant. The second equation,

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\frac{1}{\rho}\nabla p + \mathbf{g}, \tag{2}$$

can be derived from Newton's second law. The problem with Euler's equations is that they are only applicable for non-viscous fluids, which is not very practical. One of the subtler details is that the force experienced by a fluid element is always normal to its surface. However, due to the fluid elements interacting, tangential forces arise. Cauchy's equation,

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u}\right) = \nabla \cdot \mathbf{T} + \rho\mathbf{g}, \tag{3}$$

adds a stress tensor to handle these additional forces. If we choose $T_{ij} = -p\delta_{ij}$ we simply get (2), as expected, however, if we choose a more complex tensor with a deformation term, such as

$$T_{ij} = -p\delta_{ij} + \mu\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right),$$

(3) reduces to

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \rho\mathbf{g} \quad \text{or,}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{g}, \tag{4}$$

which along with the incompressibility condition, (1), are the Navier-Stokes equations. Notice that if $\nu = 0$, there is no interaction between fluid elements, and this reduces simply to Euler's equations, (2).

2.2 Reynolds-Averaged Navier-Stokes (RANS) Equation

The Navier-Stokes equations are notoriously difficult to solve since it is a second order partial differential equation, with three components, and four dependencies each. To make solving the Navier-Stokes equations easier, especially for turbulent flows, we can assume the flow is the superposition of the laminar, and turbulent flows, and then take the time average. We can make the following variable substitutions:

$$\alpha_i \equiv \overline{\alpha_i} + \alpha_i',$$
$$\text{where } \overline{\alpha_i} = \frac{1}{\tau}\int_0^\tau \alpha_i dt$$

is the steady time average on some time scale τ, and α_i' is the turbulent perturbations. It follows from this definition that

$$\overline{\alpha_i'} = 0, \quad \frac{\partial\overline{\alpha_i}}{\partial t} = 0, \quad \overline{\overline{\alpha_i}} = \overline{\alpha_i}, \quad \text{and} \quad \overline{\frac{\partial\alpha_i}{\partial x_j}} = \frac{\partial\overline{\alpha_i}}{\partial x_j}. \tag{5}$$

We can write (4) in the more suggestive component form:

$$\sum_j \frac{\partial u_i}{\partial t} + u_j\frac{\partial u_i}{\partial x_j} = \sum_j -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu\frac{\partial^2 u_i}{\partial x_j^2} + g_i,$$

then, making the appropriate substitutions, taking the time average, and using the properties from (5), we obtain

$$\sum_j \overline{u_j}\frac{\partial \overline{u_i}}{\partial x_j} + \overline{u'_j \frac{\partial u'_i}{\partial x_j}} = \sum_j -\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{u_i}}{\partial x_j^2} + \overline{g_i}.$$

Rewriting this in vector form gives the Reynolds-Averaged Navier-Stokes equation,

$$(\overline{\mathbf{u}} \cdot \nabla)\,\overline{\mathbf{u}} + \overline{(\mathbf{u}' \cdot \nabla)\,\mathbf{u}'} = -\frac{1}{\rho}\nabla \overline{p} + \nu \nabla^2 \overline{\mathbf{u}} + \overline{\mathbf{g}}. \tag{6}$$

2.3 Spalart-Allmaras (SA) Turbulent Model

While (6) makes fluid flow predictions easier, a model is still needed for ν. According to the Boussinesq hypothesis [4], an increase in the viscosity gives the effect of turbulence, and therefore $\nu = \overline{\nu} + \nu'$. One such way to model this altered viscosity and the turbulence, is the Spalart-Allmaras turbulence model. This model determines the turbulent viscosity using the following:

$$\nu' = \hat{\nu} f_{v1}; \qquad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}; \qquad \chi \equiv \frac{\hat{\nu}}{\overline{\nu}}. \tag{7}$$

Assuming the flow is incompressible, and the diffusivity is constant, $\hat{\nu}$ is obtained by solving the convection-diffusion equation,

$$\frac{\partial \hat{\nu}}{\partial t} = D\nabla^2 \hat{\nu} - \overline{\mathbf{u}} \nabla \hat{\nu},$$

where D is the diffusivity.

3 Stanford University Unstructured (SU^2) Code

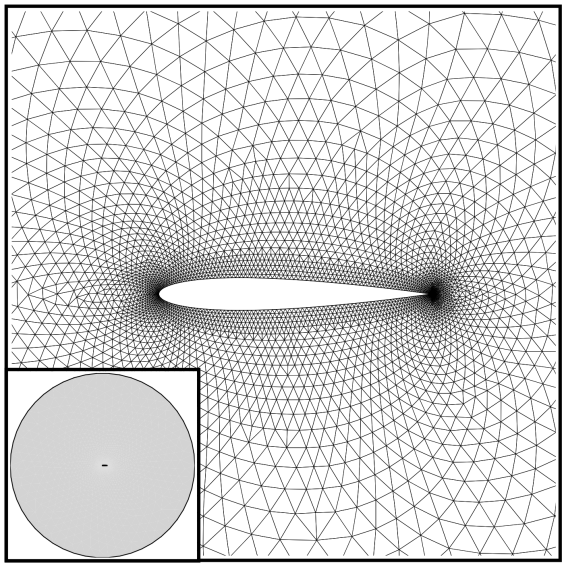


Figure 2: Unstructured mesh of an airfoil [7].

The Stanford University Unstructured (SU^2) code[1] is a relatively new, open source, constrained partial differential equation solver, optimized for computational fluid dynamics problems. It was initially developed by Francisco Palacios, Thomas Economon, and Juan Alonso from the Aerospace Design Laboratory of the Department of Aeronautics and Astronautics at Stanford University. SU^2 is written mostly in C++; and because it is open source, the capabilities have since been extended by many teams of developers from around the world, in addition to the team at Stanford University.

SU^2 was initially developed under the philosophy that it should remain open source, portable, and highly encapsulated while operating at a high performance standard. The code uses only widely available libraries and dependencies for C++ and Python; furthermore, it is written at a very high, abstract level to minimize disc space, and maximize performance. This philosophy was used to eliminate many of the problems other computational fluid dynamics software are plagued with such as price, compatibility, and memory [1]. Some of the features of the SU^2 code include, but are not limited to:

[1] `http://su2.stanford.edu/`

- Reynolds-Averaged Navier-Stokes equations (6)
- Spalart-Allmaras turbulence model (7) ($c_{v1} = 7.1$ [1] [2])
- Menter Shear Stress Transport model (an alternative to the SA model)
- Compressible flows
- Wave equation, heat equation, and plasma equations
- Optimal shape design (i.e. Supersonic aircraft)
- Dynamic meshes

To conduct a simulation, two main files are required, the configuration file, and the mesh file. All of the parameters such as the fluid speed, Reynolds number, and governing equations are specified in the configuration file. The shape geometry of the problem is stored as an unstructured mesh – hence the name – in the mesh file, an example visualization is in Figure 2. The mesh is unstructured in the sense that there is no algebraic relation between nodes. Instead, the mesh is specified by listing the node coordinates, and the adjacent nodes it is connected to. The density of nodes is greatest at the surface of the object since it has the most complex fluid flow, and any errors could quickly propagate through the solution.

Typically a particular node within the mesh will be surrounded by six triangular cells. Using the centroids of the cells, as well as the midpoints of the edges, we can construct a hexagonal region, Ω_i, which surrounds the ith node. Our partial differential equations can be discretized by

$$\int_{\Omega_i} \frac{\partial \mathbf{u}}{\partial t} d\Omega + R_i(\mathbf{u}) = 0,$$

where $R_i(\mathbf{u})$ is the residual. After the spacial components have been discretized, the governing equations can be numerically solved with a variety of methods and optimizations. Furthermore, due to the way the solution is discretized, the mesh can dynamically be updated; this is a key feature in the shape optimization process of SU^2.

4 Scripting and Automation

Throughout this project Wolfram Mathematica 10.1.0 for Linux x86 was used to sift through the vast data files and extract the relevant information to produce the plots. The SU^2 code natively writes to the data files in the format used by Tecplot[2], a visualization and analysis tool for computational fluid dynamics. Since Tecplot was not used, a Wolfram function was needed to convert the data to a form Mathematica could use. Once the data was in a usable form, the `ListDensityPlot` function was used to create the images. The function to tidy the data along with the plotting function were combined into a Wolfram script such as Figure 3, which can be executed from the terminal. To fully automate the image generation, a shell script was written to iterate over each data file.

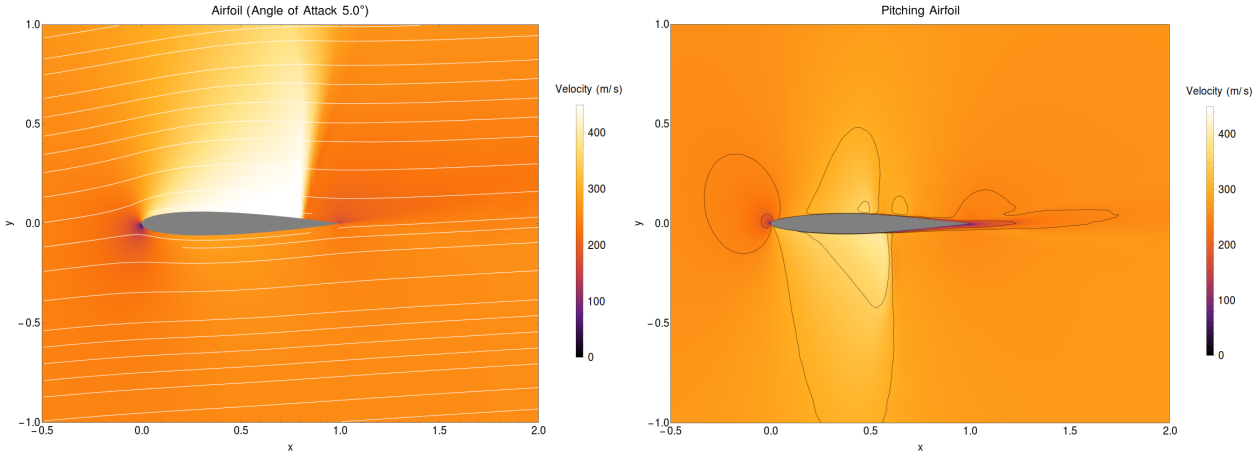
[2] `http://www.tecplot.com/`

```
#!/usr/local/bin/WolframScript -script

(*Change the current directory*)
SetDirectory["/home/brady/SU2/CFD/Results/Pitching_Airfoil_Turb"];
Print[ToString[$CommandLine[[4]]]]; (*To check the progress*)

CSV = "surface_flow_0" <> ToString[$CommandLine[[4]]] <> ".csv";
DAT = "flow_0" <> ToString[$CommandLine[[4]]] <> ".dat";
PNG = "Airfoil_Turb" <> ToString[$CommandLine[[4]]] <> ".png";

(*Set the plot limits, colour function, and the legend style*)
xyzlimits = {{-0.5, 2},{-1, 1},{0, 450}};
colfunc = ColorData["SunsetColors"][#/xyzlimits[[3,2]]] &;
leg = BarLegend[{colfunc, xyzlimits[[3]]}, LegendLabel → "Velocity␣(m/s)",
    LegendMarkerSize → 500];

(*Draw a gray ploygon using the points of the surface_flow.csv*)
shape = Graphics[{Gray, Polygon[Import[CSV][[2;;-1, {2, 3}]]]}];

(*Clean the data so it's in a usable form*)
(*Import the data file, and remove the preamble (three lines)*)
datafile = Import[DAT][[4 ;; -1]];

(*There is four seemingly random numbers per line for several lines at*)
(*the end, this ignores those lines*)
Do[If[Dimensions[datafile[[i]]][[1]] == 4,
   {CleanData = datafile[[1 ;; i - 1]], Break[]}],
   {i, 1, Dimensions[datafile][[1]]}];
(*Only the first 5 columns are needed: x, y, \[Rho], \[Rho]u, \[Rho]v*)
Data = CleanData[[All, 1 ;; 5]];

(*Declare and fill array for the velocity*)
velocity = {};
Do[AppendTo[velocity, {Data[[i, 1]], Data[[i, 2]],
   Sqrt[(Data[[i, 4]]/Data[[i, 3]])^2 + (Data[[i, 5]]/Data[[i, 3]])^2]}],
   {i, 1, Length[Data]}]

velplot = ListDensityPlot[velocity, ColorFunction → colfunc,
   PlotRange → xyzlimits, AspectRatio → Automatic,
   LabelStyle → {Black, FontSize → 18}, PlotLegends → leg,
   ColorFunctionScaling → False, Frame → True,
   FrameLabel → {"x", "y"}, PlotLabel → "Pitching␣Airfoil",
   ImageSize → Full];

contplot = ListContourPlot[velocity, PlotRange → xyzlimits,
   ContourShading → None, Contours → {200, 250, 300, 350}];

SetDirectory["/home/brady/SU2/CFD/TeX/Airfoil_Animation_Turb"];
Export[PNG, Show[velplot, contplot, shape]]
```

Figure 3: The Wolfram script used to generate Figure 4b. Syntax highlighting modified from [5].

5 Results

5.1 Airfoil



(a) Laminar Euler simulation of an airfoil.

(b) Turbulent Navier-Stokes simulation of a pitching airfoil, with velocity contours.

Figure 4: Results of the airfoil simulations.

A NACA 0012 airfoil is the first tutorial in the Quick Start section of SU^2's documentation [7]. This series of airfoils was developed by the National Advisory Committee for Aeronautics (NACA); for this particular case 0012 is the code for a symmetric airfoil with a thickness 12% of its length. As this is the first tutorial, it walks the user through the process of configuring, and running a simulation in SU^2 for the first time after it is installed – a natural place to start.

	5.0° Angle	Pitching
Method	Euler	Navier-Stokes
Turbulence	None	Spalart-Allmaras
Flow Speed	270 ms^{-1}	270 ms^{-1}
Reynolds Number	0	12560000
Temperature	300.0 K	300.0 K
Pressure	1 atm	N/A
Iterations	250	25 per frame

Table 1: Parameters used in the airfoil simulations. Results shown in Figure 4.

In the first simulation Euler's equations were used to simulate an airfoil during takeoff. The full specifications of both simulations can be found in Table 1. Figure 4a shows the streamlines, and velocity of the air around the airfoil from the data. It is clear from the data that due to the shape of the airfoil, the velocity of the air that flows over the wing of an aircraft is greatly increased compared to the air that flows under. A consequence of this increase in velocity is a decrease in pressure, which in turn provides lift for the aircraft.

The second simulation was of a pitching airfoil, the angle continuously varied from $-4.0°$ to $4.0°$ – unlike the first simulation where the angle was a constant $5.0°$. Furthermore, the Navier-Stokes equations with turbulence were used. Figure 4b shows the results of the simulation; using a viscous turbulent flow, such as real aircraft experience, there is undoubtedly a long turbulent stream off of the tail end of the airfoil. Another interesting property is that some of the velocity contours initially form in two distinct areas, and they come together as the airfoil flattens out.

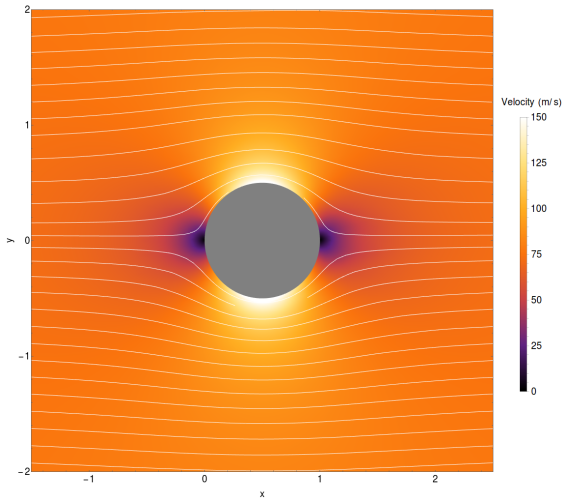
5.2 Laminar Flow Around a Static Cylinder

In our second simulation, we sought out to compare how the different governing equations, Euler's and Navier-Stokes, effects the flow past a static cylinder. The key difference in the two parts of the simulation

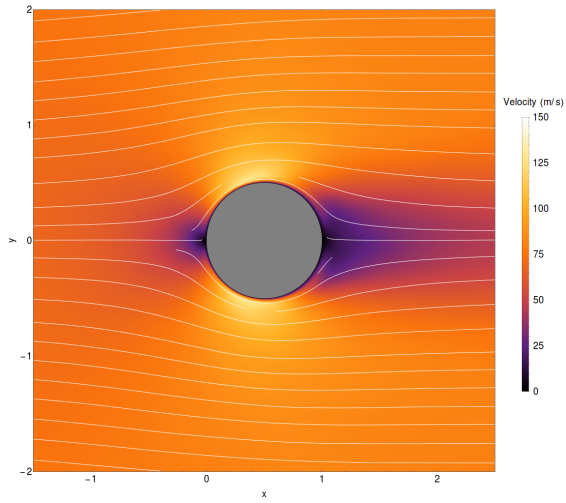
was the viscosity, the Euler simulation was inviscid, whereas the Navier-Stokes simulation used a Reynolds number of 20000, the full details can be found in Table 2, and the results are shown in Figure 5.

A thought-provoking property of the Euler simulation (Figure 5a) is that the results are completely symmetric, both the streamlines, as well as the velocity profile. While this agrees with theory, it is still fascinating that the direction of flow cannot be determined. Another interesting property to note is the lack of a boundary layer (Figure 5c). On the boundary of the cylinder for a non-viscous fluid, the fluid has the greatest velocity at the boundary, and gradually decreases as the distance to the boundary increases.

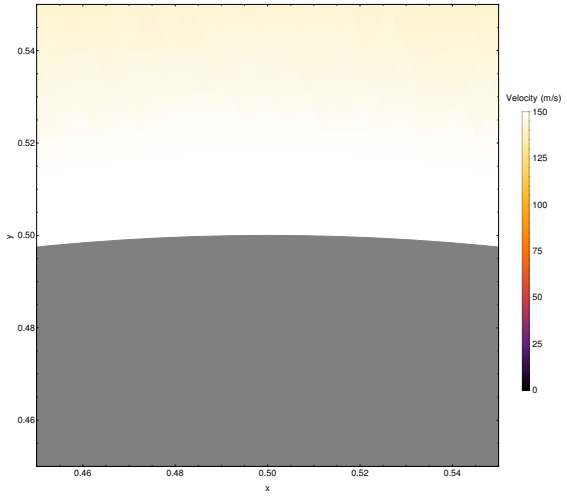
Conversely, we see neither of these properties for a viscous fluid. We instead observe a tail behind the cylinder of reduced velocity (Figure 5b), as the fluid crashes into the front of the cylinder it is pushed outward, creating a region of very low velocity. Furthermore, it is clear in Figure 5d that a boundary layer forms on the surface of the cylinder. This is a characteristic of all viscous flows, and is known as the no-slip condition. The fluid against the boundary gets 'stuck' and has a velocity of zero. Outside the boundary layer we observe a similar situation to the Euler simulation except the peak velocity is no longer in the area of the cylinder that is perpendicular to the flow, it is shifted leftwards toward the incoming fluid.



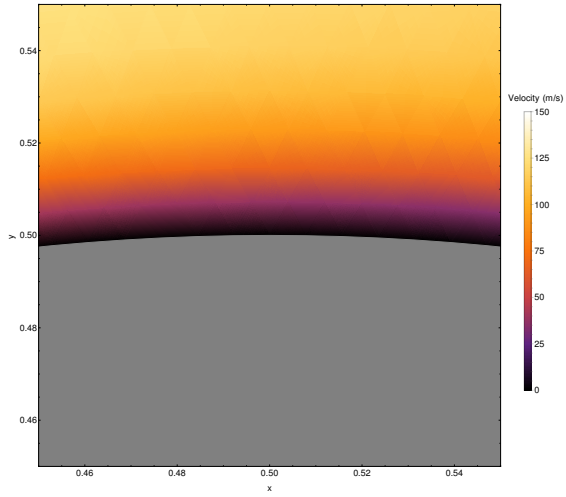
(a) Euler's equation simulation results.



(b) Navier-Stokes equations simulation results.



(c) No boundary layer forms for an inviscid fluid, and the flow speed is the fastest at the boundary.



(d) For a viscous fluid, a boundary layer is formed and the flow speed quickly decreases to zero.

Figure 5: Results of the static cylinder experiments.

5.3 Turbulent Flow Around a Static Cylinder – Vortex Shedding

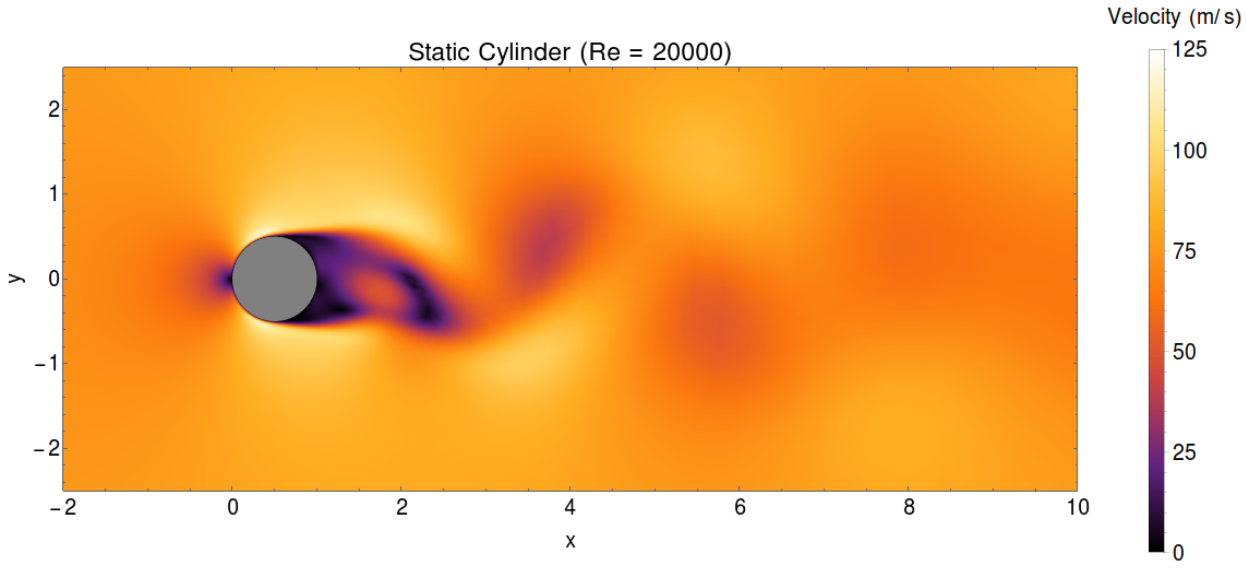


Figure 6: Animation of the vortex shedding. The full video can be found at `https://youtu.be/1lBQNlEdXss`.

Our final experiment was similar to the static cylinder simulation, however, using the Spalart-Allmaras turbulence model. The Reynolds number was the same as the prior simulation, and a time step of 0.0006s was used, the remaining configuration options are shown in Table 3. Initially, we obtained the same results as in Figure 5b, as expected. However, around the 200th iteration, or 0.1200s, the first vortex is shed. Following this, vortices were alternatingly shed from the top and bottom of the back of the cylinder, and were swept away by the fluid, allowing a new vortex to be formed. An animation of this process can be seen in Figure 6.

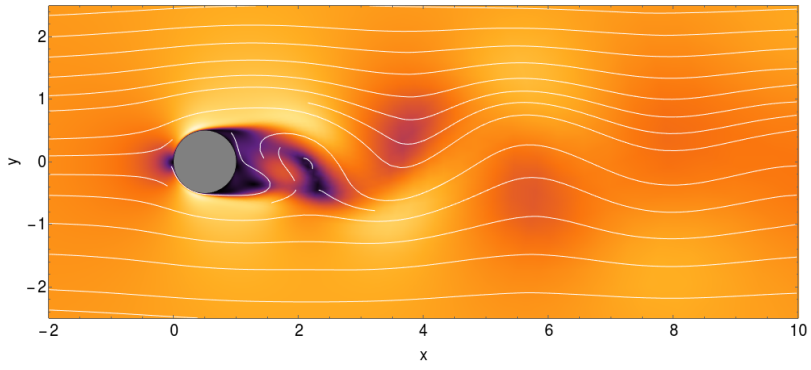
The von Kármán vortex street is initiated by perturbations in the steady state solution from the turbulence. The vortices are periodically shed in the same fashion to Figure 6 for the remainder of the simulation, a total of 2000 iterations. The period of the vortex shedding is 112 iterations, which implies $T = 0.0672$ s, and $f = 14.88$ Hz. The formation of the vortices can more easily be seen in Figure 7.

Method	Euler	Navier-Stokes
Turbulence	None	None
Flow Speed	75 ms^{-1}	75 ms^{-1}
Reynolds Number	0	20000
Temperature	300.0 K	300.0 K
Pressure	1 atm	N/A
Iterations	250	250

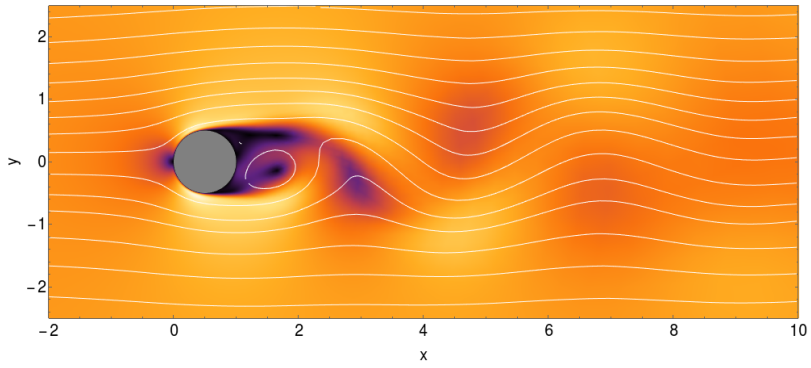
Table 2: Parameters used in the laminar flow simulations. Results shown in Figure 5.

Method	Navier-Stokes
Turbulence	Spalart-Allmaras
Flow Speed	75 ms^{-1}
Reynolds Number	20000
Temperature	300.0 K
Time Step	0.0006 s
Iterations per Time Step	10
Total Iterations	2000
Total Time	1.200 s

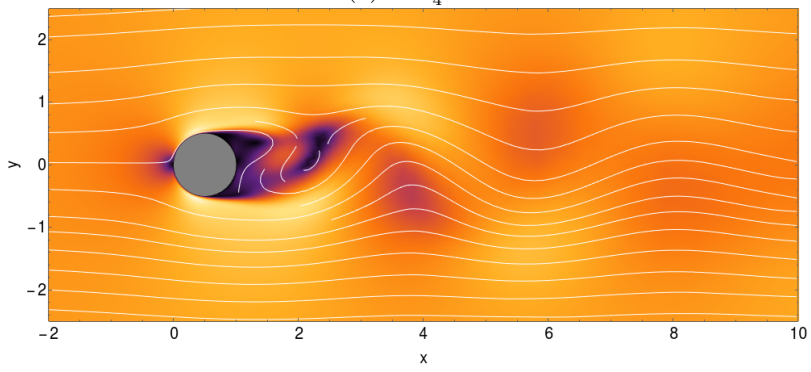
Table 3: Parameters used in the turbulent flow simulations. Results shown in Figures 6, and 7.



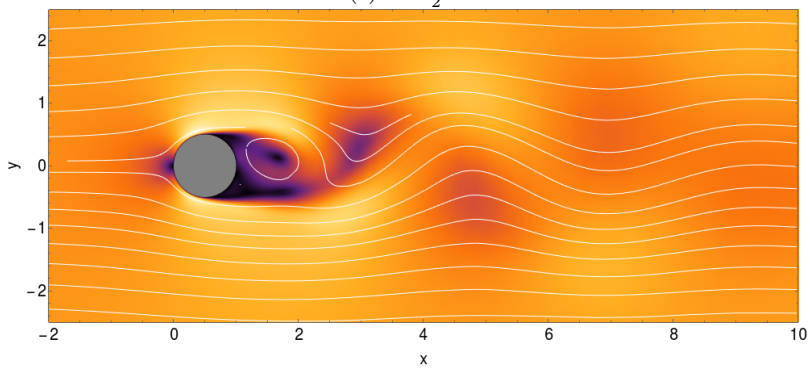
(a) $t = 0$



(b) $t = \frac{1}{4}T$



(c) $t = \frac{1}{2}T$



(d) $t = \frac{3}{4}T$

Figure 7: Vortex shedding in increments of $\frac{1}{4}T$.

6 Conclusion

This project examined the theory behind fluid dynamics, as well as numerical simplifications that can be made to optimize the computation. To acquire the visualizations, a Wolfram script along with Wolfram Mathematica was used to parse the data files and create the graphics. The preliminary results in this project are in agreement with the theoretical predictions. SU^2 is a viable computational fluid dynamics simulator – however, only a small set of its capabilities were used in this project. We were able to simulate a von Kármán vortex street behind a static cylinder using the Navier-Stokes equations, and the Spalart-Allmaras turbulence model. Computational fluid dynamics is now a critical discipline of fluid dynamics, and it is only going to become more indispensable as computations become faster, and the numerical algorithms become further optimized.

References

[1] Francisco Palacios, Juan Alonso, Karthikeyan Duraisamy, Michael Colonno, Jason Hicken, Aniket Aranake, Alejandro Campos, Sean Copeland, Thomas Economon, Amrita Lonkar, Trent Lukaczyk, and Thomas Taylor, *Stanford University Unstructured (SU^2): An open-source integrated computational environment for multi-physics simulation and design*, 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, 2013.

[2] Francisco Palacios, Thomas D. Economon, Aniket Aranake, Sean R. Copeland, Amrita K. Lonkar, Trent W. Lukaczyk, David E. Manosalvas, Kedar R. Naik, Santiago Padron, Brendan Tracey, Anil Variyar, and Juan J. Alonso, *Stanford University Unstructured (SU^2): Open-source Analysis and Design Technology for Turbulent Flows*, 52nd Aerospace Sciences Meeting. National Harbor, Maryland, 2014.

[3] John D. Anderson, Jr., *Computational Fluid Dynamics*, McGraw-Hill, Inc., 1995.

[4] D.C. Wilcox, *Turbulence Modeling for CFD*. 2nd Ed., DCW Industries, Inc., 1998.

[5] Belisarius (`http://mathematica.stackexchange.com/users/29581/belisarius`), *Highlighting Mathematica code in LaTeXdocument*, URL: `http://mathematica.stackexchange.com/a/83811`

[6] yo' (`http://tex.stackexchange.com/users/11002/yo`), *How to make a combined List of Figures and Tables?*, URL: `http://tex.stackexchange.com/a/258508`

[7] `https://github.com/su2code/SU2/wiki/Quick-Start`

[8] `http://news.stanford.edu/news/2012/january/aero-engineering-software-012412.html`