Math 361S Spring 2024: Final Project
Using Gauss-Seidel for a basic CFD

Dadmehr Ghasemfar and Derrick Roseman

April 26, 2024

Abstract

Computational Fluid Dynamics (CFD) is a branch of fluid mechanics that uses numerical methods and algorithms to solve and analyze problems that involve fluid flows. This technology is widely applied in various industries such as aerospace, automotive, energy, and environmental engineering to optimize designs, predict performance, and understand complex flow phenomena without the need for physical prototypes. However, CFD simulations have several limitations. Commonly, CFD simulations can be very computationally expensive and time-consuming. This limits its potential in applications and brings a need for a faster form of CFD. This is where the Fast Fluid Dynamics (FFD) algorithm comes in. In this paper we first derive the fundamental equations for fluid mechanics, then explain the working principle of the FFD algorithm, and then demonstrate the application of this method in a simple program. To simplify the task, we only model incompressible and inviscid flow.

Contents

1 Mathematical Background

To simulate fluid behaviour with the FFD algorithm we must first derive the equations describing fluid mechanics - the **Navier Stokes equations**. For this we must define certain mathematical tools [1].

In the remainder of this discussion **V** is a fluid control volume and **S** is the surface of this volume. $u(\mathbf{x}, t)$ is the velocity vector describing the flow at position $\mathbf{x}$ and time t (units of $\frac{m}{s}$). Similarly, $P(\mathbf{x}, t)$ is the pressure at $\mathbf{x}$ at time t (units of Newton-seconds). Moreover ρ is the fluid density (in $\frac{kg}{m^3}$), M is the total system mass, $\overline{\mathbf{p}}$ is the total system momentum, and $\hat{n}$ is the surface normal vector at a particular point.

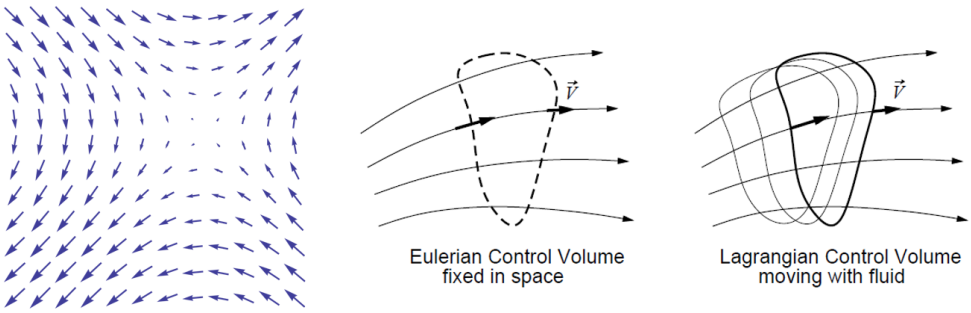


Figure 1: An example vector field and diagram representations of Eulerian and Lagrangian fluid systems. Vector fields are used to represent fluid velocities in a region. Operations on the vector field represent operations on the flow.

There are two fundamental ways to mathematically model fluids. The **Eulerian perspective** examines a fixed region in space and looks at the fluid entering and leaving. The Lagrangian perspective examines a volume of fluid, tracking it as it moves in space. In simulation, a Lagrangian method corresponds to a particle based simulation while an Eulerian method corresponds to an FEA-style cell-by-cell based simulation. We will be focusing on the Eulerian perspective for this work.

Traditional partial time derivatives $\frac{\partial \overline{\mathbf{u}}}{\partial t}$ do not accurately model vector field changes with respect to time. This is because two effects are simultaneously at work. The vector field at a point is changing with respect to time, but the vector field itself also moves along the flow as a function of time. Say a fluid element moves from point A to point B in the time t to t+1. The velocity $\overline{\mathbf{u}}(B, t+1)$ is not just the velocity at $\overline{\mathbf{u}}(B, t)$ plus some derivative term. It is in fact $\overline{\mathbf{u}}(A, t)$ plus some derivative term. This process is called **advection**.

To account for this, the idea of the partial derivative is extended to the **material derivative**, also known as the **advective derivative**. This is defined for any scalar or vector y as:

$$\frac{Dy}{Dt} = \frac{\partial y}{\partial x} + \overline{\mathbf{u}} \cdot \nabla y \tag{1}$$

We know that for a particular fluid element two things can effect system mass. Either the density of the flow changes (due to time related change and advective change, the material derivative $\frac{D\rho}{Dt}$ covers both) and/or mass enters or leaves the system at the boundaries. For an infinitesimal fluid element, the overall flow into or out-of the element is the divergence of the flow field at that point times the density

of the flow $\rho(\nabla \cdot \overline{\mathbf{u}})$. Hence, the overall mass change of an entire region is simply the integral of these two terms over the region. Setting this to zero we get the **mass conservation equation** and it can be slightly simplified and shown in a differential form as well. Both are shown below:

$$\frac{DM}{Dt} = \iiint_R \frac{D\rho}{Dt} + \rho(\nabla \cdot \overline{\mathbf{u}})dV = 0 \tag{2}$$

$$\frac{\partial \rho}{\partial x} + \nabla \cdot (\rho \overline{\mathbf{u}}) = 0 \tag{3}$$

Now instead of examining the change in mass of the system, let us examine the change in momentum of the system. Each element has a momentum vector, the momentum of the region is the integral of the element momentum across the region. As before two things can change the momentum of an element.

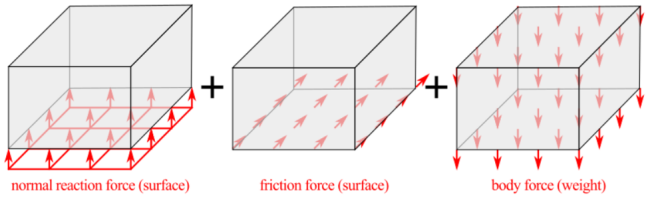


Figure 2: The three relevant fluid forces on a 3D fluid volume. Normal and shear forces act on the boundaries while body forces act at all points. Normally, normal forces represent pressure, shear forces represent viscosity, while body forces represent gravity.

If mass flows across the boundary with some velocity, that element gains or losses momentum. For a given boundary, the momentum change can be described as $\overline{\mathbf{p}} = \dot{m}\overline{\mathbf{u}} = \rho(A\hat{n} \cdot \overline{\mathbf{u}})\overline{\mathbf{u}}$.

From classical physics we know that momentum change happens when a force is applied. These forces can be surface normal forces (also known as **hydrostatic forces**), surface shear forces (also known as **viscous forces**), and finally **body forces**. Integrating the momentum transfer across the boundary and the momentum change in the volume, we get:

$$\iiint_R \rho\overline{\mathbf{u}}dV + \oiint_S \rho\overline{\mathbf{u}}dA = \frac{d\overline{P}}{dt} + (\overline{P}_{out} - \overline{P}_{in}) = \overline{F}_{applied} = \oiint_S -P\hat{n}dA + \iiint_R \rho\overline{g}dV \tag{4}$$

We can use the generalized Stokes Theorem to rewrite all of the 2D surface integrals as 3D volume integrals. With everything being integrated over R, we then set the entire integral equal to zero, since overall momentum should not change. Since the volume integral is always equal to zero, it must be zero everywhere. Finally, divide both sides by ρ to get the most important equation for fluid flow, the **momentum conservation equation**.

$$\frac{\partial \overline{\mathbf{u}}}{\partial t} + (\overline{\mathbf{u}} \cdot \nabla)\overline{\mathbf{u}} = -\frac{1}{\rho}\nabla\overline{\mathbf{P}} + \overline{g} + \overline{F}_{viscous} \tag{5}$$

When expanded out (replacing $\overline{\mathbf{u}}$ with its components (u, v, w)), this equation becomes three partial differential equations - one for each dimension. Here we have left the viscous forces as their separate term. This is because we are assuming inviscid flow where viscous forces are all zero.

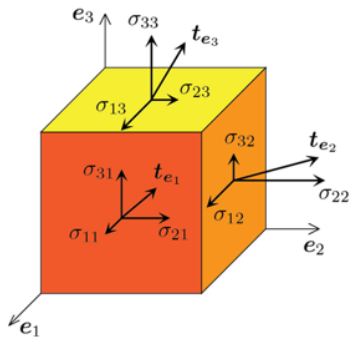
$$\frac{\partial(\rho u)}{\partial t} + \nabla \cdot \left(\rho u \vec{V}\right) = -\frac{\partial p}{\partial x} + \rho g_x + (F_x)_{\text{viscous}}$$

$$\frac{\partial(\rho v)}{\partial t} + \nabla \cdot \left(\rho v \vec{V}\right) = -\frac{\partial p}{\partial y} + \rho g_y + (F_y)_{\text{viscous}}$$

$$\frac{\partial(\rho w)}{\partial t} + \nabla \cdot \left(\rho w \vec{V}\right) = -\frac{\partial p}{\partial z} + \rho g_z + (F_z)_{\text{viscous}}$$

Figure 3: The momentum conservation equation, written in differential form and split by dimensions (x,y,z).

Assuming the fluid does have viscous shear forces, there is still a simple way to calculate the $\overline{\mathbf{F}}_{viscous}$ term. For cubic fluid elements, decompose the total force into the normal and shear components in each of the three major planes. Rewriting these components in tensor form, we get the **Cauchy stress tensor** σ. The **mean hydrostatic stress** π is then defined as the average of the diagonal components. In other words, the average normal force on the element is $\pi = \frac{1}{3}(a_{11} + a_{22} + a_{33})$. What is left behind when you subtract this average normal stress is called the **deviatoric stress tensor** τ. This tensor describes all shear forces at a point. Each point of the fluid will have its own corresponding tensor. The viscous force $\overline{\mathbf{F}}_{viscous}$ is then just the divergence of this tensor over the field.



$$\begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \quad \begin{bmatrix} \sigma_{11} - \pi & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} - \pi & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} - \pi \end{bmatrix}$$

Figure 4: To the left is a cubic fluid element with all applied forces decomposed by location and direction. The σ tensor is all those components in a tensor. The deviatoric stress tensor τ is shown to the right.

This concludes the fundamental understanding of fluid mechanics needed to understand the FFT algorithm. Given the mass conservation and momentum conservation equations, the flow field can be uniquely solved given boundary conditions, initial conditions, and all applied forces. Because we are only interested in incompressible flows (e.g water), we utilize one more theorem - Helmoltz Decomposition.

The fundamental theorem of vector calculus states that any smooth and rapidly decaying vector field in 3D can be resolved into the sum of an irrotational (curl-free) vector field and a solenoidal (divergence-free) vector field. The process of splitting a given field into these two parts is called **Helmholtz decomposition**. Though we will not be proving this theorem, it is a key part of the FFD algorithm.

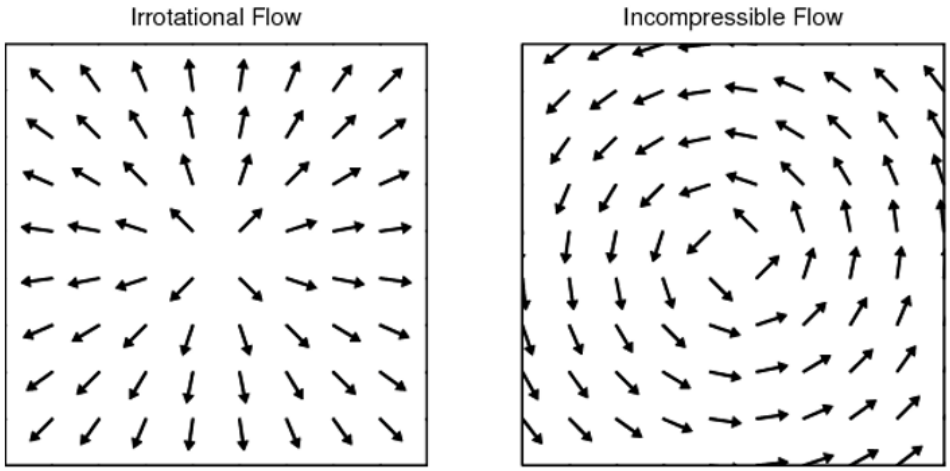


Figure 5: Two fundamental flow types. The irrotational flow to the left only has divergence. The incompressible flow to the right only has curl.

2 Fast Fluid Dynamics Algorithm

Our implementation of the FFD algorithm was created by Jos Stam in his 2001 publication titled *stable fluids*[2]. His background is in computer graphics and his task was to accelerate simple flow CFD for use in animation and video games. His algorithm takes a vector field at time t and returns the updated vector field at time t+1. Boundary values, initial conditions, and external forces are supplemental inputs. For the remainder of this discussion, boundary velocities are set to zero and gravity is the only external force.

$$\mathbf{w}_0(\mathbf{x}) \xrightarrow{\text{add force}} \mathbf{w}_1(\mathbf{x}) \xrightarrow{\text{advect}} \mathbf{w}_2(\mathbf{x}) \xrightarrow{\text{diffuse}} \mathbf{w}_3(\mathbf{x}) \xrightarrow{\text{project}} \mathbf{w}_4(\mathbf{x})$$

The process can be cleanly broken down into four operations, as shown above. Starting with $W_0 = \overline{\mathbf{u}}$, each step operates on the vector field $\mathbf{W}$ until it is a a close solution to $\overline{\mathbf{u}}$ at t+1. The order of the operations is not critical, as step one (add force) happens before step four (project). The algorithm is summarized below:

1. Add all of the given external forces to all the elements of your fluid. In our simulation, the only force being added is that of gravity.

2. Advect all fluid elements forward one time-step (see intro of *Mathematical Background* for a description of advection). In true **advection**, each fluid element velocity at time t+1 assumes the value of the fluid that would end up there given the flow field at t+1. Said differently, at point $\mathbf{x}$ replace the velocity with that which would end up at $\mathbf{x}$ given the flow at time t+1. This is computationally impossible, since we do not know the flow at t+1. Instead, we approximate by using the value of the field at time t. In other words, take the velocity of the element that currently ends up at our point of interest. To find a value between grid points, use **linear interpolation** between grid points.

3. This step captures the effect of **viscosity** and is equivalent to the diffusion equation $\frac{\partial \overline{\mathbf{W}}}{\partial t} = \nu \nabla^2 \overline{\mathbf{W}}$. For our application, since we are assuming inviscid flow, this step is ignored.

4. This step is the most important in regards to the focus of this project. After steps 1-3, the resulting W_3 vector field will not have a zero divergence everywhere. This step **forces incompressibility**. Let $\mathbf{P}()$ be the operation that keeps all but the divergent component of a vector field. In other words, for an incompressible flow $\mathbf{P}(\overline{\mathbf{u}}) = \overline{\mathbf{u}}$ and since the pressure field p is only divergent and completely irrotational $\mathbf{P}(p) = 0$. Now if $\mathbf{W} = \mathbf{u} + \nabla \mathbf{q}$ (incompressible and irrotational components), then $\mathbf{P}(\mathbf{W}) = \overline{\mathbf{u}}$ is the divergent-free part of $\mathbf{W}$, then $\nabla \cdot \mathbf{W} = \nabla \cdot \mathbf{u} + \nabla \cdot \nabla \mathbf{q} = \nabla^2 \mathbf{q} = \Delta \mathbf{q}$. It may seem like we have just rephrased, but the equation is now a classic 3D Poisson equation for the scalar field $\mathbf{q}$ with the Neumann boundary condition. Solutions to this can be rapidly computed by solving a sparse tridiagonal matrix using **Gauss-Seidel** [4]. Applying this operator to the momentum equation and using the stated properties, we get $\frac{\partial \overline{\mathbf{u}}}{\partial t} = \mathbf{P}(-(\overline{\mathbf{u}} \cdot \nabla)\overline{\mathbf{u}}) + \nu \nabla^2 \overline{\mathbf{u}} + \mathbf{f})$.

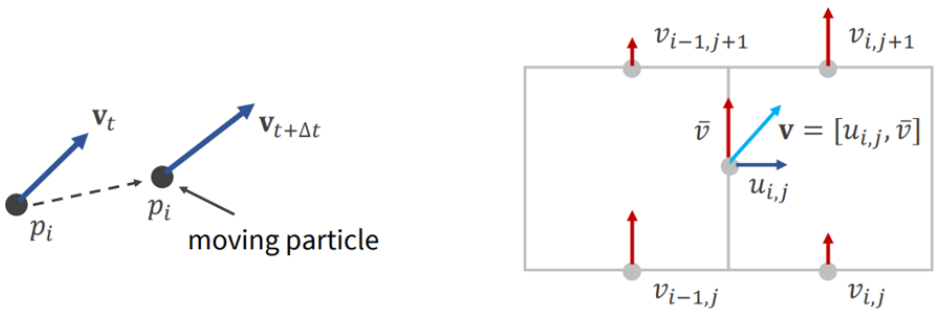


Figure 6: A diagram representation of fluid advection, as well as the method for finding the average velocity at an x-indexed point of the collocated grid. This averaged vector is what is traced back in time when implementing the advection step.

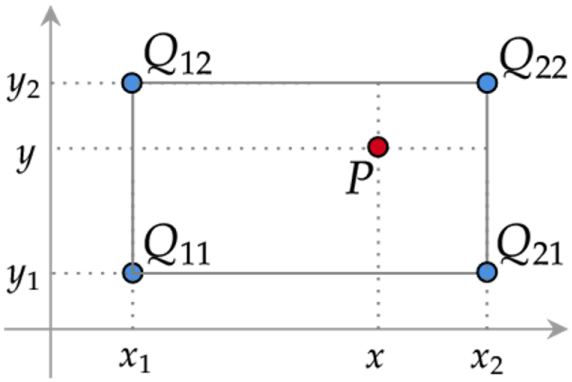


Figure 7: A diagram representation of 2D linear interpolation between known grid points. The value at P will be a linear sum of the values of the grid points. Each point will be weighted by the corresponding area in the rectangle seen above.

3 Implementation and Gauss-Seidel

As we showed above, solutions to the projection step (and in our case the unused diffusion step) are simply solutions to some sparsely filled matrix. Details regarding the derivation of this fact can be found in (Zhang, 1998)[4]. Fortunately, we have a fast and efficient technique from MATH361 that can solve sparse matrices - the Gauss Seidel iterative method.

All iterative methods attempt to solve the matrix equation $\mathbf{A}\overline{x} = \mathbf{b}$, for some known $\mathbf{A}$ and $\mathbf{b}$. This is done by breaking up $\mathbf{A}$ into the sum $\mathbf{A} = \mathbf{M} - \mathbf{N}$ and then rewriting to solve for $\overline{x}$. This gives:

$$x^{k+1} = \mathbf{M}^{-1}\mathbf{N}x^k + \mathbf{M}^{-1}\mathbf{b} \text{ where } x^{k+1} \text{ is the k+1 iterated solution of x} \tag{6}$$

The Gauss-Seidel method is an iterative technique used to solve systems of linear equations, particularly when they arise from finite difference or finite element discretizations of partial differential equations. In the format shown above, we get $\mathbf{M} = \mathbf{L}_A + \mathbf{D}_A$ and $\mathbf{N} = \mathbf{U}_A$ for Gauss-Seidel (where $\mathbf{L}_A, \mathbf{D}_A, \mathbf{U}_A$ are the lower, diagonal, and upper components of $\mathbf{A}$ respectively). It is named after German mathematicians Carl Friedrich Gauss and Philipp Ludwig Von Seidel. Here's how it works:

Initialization: Start with an initial guess for the solution vector. Iteration: At each iteration, update each component of the solution vector using the most recent values of the other components. Repeat the iteration until the solution converges to the desired accuracy.

Theorem 2.1. *Let* $\rho(\boldsymbol{A}) = \max_i |\lambda_i|$ *be the spectral radius. Then*

$$\rho(\boldsymbol{A}) = \inf_{\|\cdot\|} \|\boldsymbol{A}\|,$$

where the infimum is taken over all induced matrix norms. In particular, it follows that $\rho(\boldsymbol{A})$ *is the 'smallest' norm in the sense that*

(i) If $\|\cdot\|$ *is any induced matrix norm then* $\rho(\boldsymbol{A}) \leq \|\boldsymbol{A}\|$.

(ii) If $\rho(\boldsymbol{A}) < 1$*, then there is an induced matrix norm for which* $\rho(\boldsymbol{A}) < \|\boldsymbol{A}\| < 1$ *(i.e., a norm that, applied to* $\boldsymbol{A}$*, is slightly bigger than* $\rho(\boldsymbol{A})$*).*

Figure 8: Notes taken from the MATH361 lecture given on iterative methods [5]

Gauss-Seidel fails to converge when the spectral radius of the iteration matrix is greater than one. Furthermore, it converges rather slowly when the spectral radius of the iteration matrix is close to one. For this reason, overrelaxation is implemented. Overrelaxation is a method that speeds up convergence of Gauss-Seidel. In our case, overrelaxation is simply implemented by adding a constant term in front of the calculated divergence in the projection step. If this constant term is chosen between one and two, convergence is sped up significantly. In our code, we found that values of 1.7-1.8 for the constant term work best.

To implement Gauss-Seidel on $\mathbf{W}_3$ (vector field before projection step) to solve $\mathbf{W}_4$ (vector field after

projection), we iterate according to the pseudocode shown.

$$d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$$

$$s \leftarrow s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}$$

$$\begin{aligned}
u_{i,j} &\leftarrow u_{i,j} + d\, s_{i-1,j}/s \\
u_{i+1,j} &\leftarrow u_{i+1,j} - d\, s_{i+1,j}/s \\
v_{i,j} &\leftarrow v_{i,j} + d\, s_{i,j-1}/s \\
v_{i,j+1} &\leftarrow v_{i,j+1} - d\, s_{i,j+1}/s
\end{aligned}$$

Figure 9: Here **d** refers to the divergence of the vector at coordinates (i, j). Some are velocity components are subtracted since the left-face normal points in the negative x-direction (similar argument for y), while others are added. Furthermore, **S** refers to an elements number of neighboring fluid elements. If an element has two neighboring obstacles, S) = 2. If it has three neighboring obstacles, S = 1. With four neighboring obstacles, S=0. Similarly, **s(i,j)=1 if (i,j) is a fluid element** and 0 if it is an object. The **v** field is simply our velocity flow field.

Below are a few screenshots of code to help understand the implementation of the FFD algorithm. The first screenshot shows the first step where the body force, gravity, is added. We iterate through all points and add gravity. We then force all boundary velocities and obstacle velocities to zero. This is called the no-slip and no-penetration boundary condition. The second screenshot shows the next step - advection. In this step, the velocity field at a location is updated by looking at the velocity field a small distance step behind and making that the new velocity field. For this step, a function called "sampleField" is called.

```
for x = 1:n_x
    for y = 1:n_y
        x_pos = x_min+x*dx;
        y_pos = y_min+y*dy;

        if (objects(x, y) == 0 && objects(x, y-1) == 0)
            u(x, y, 2) = u(x, y, 2) + dt*(-9.81);
        end


        if (objects(x, y) == 1)
            u(x, y, 1) = 0;
            u(x, y, 2) = 0;
        end


        if (x == 1 && y ~= n_y && y ~= 1)
            u(x, y, 1) = 0.5;
        end
```

Figure 10: Screenshot of code depicting the body force step

```
%% ADVECTION AND POST PROCESSING
new_u = u;
for x = 2:n_x-1
    for y = 2:n_y-1

        pos_current = [x_min+x*dx, y_min+y*dy];

        if (objects(x,y) == 0 && objects(x-1, y) == 0)

            vel_x_avg_current = u(x, y, 1);
            vel_y_avg_current = 0.25*(u(x-1, y+1, 2) + u(x-1, y, 2) + u(x, y+1, 2) + u(x, y, 2));


            vel_current = [vel_x_avg_current, vel_y_avg_current]; %[u(x, y, 1), vel_y_avg_current];
            pos_prev = pos_current - dt*vel_current;

            vel_avg_x_prev = sampleField(pos_prev(1), pos_prev(2), x_min, x_max, y_min, y_max, dx, dy, u(:, :, 1));

            new_u(x, y, 1) = vel_avg_x_prev;
        end
```

Figure 11: Screenshot of code depicting the advection step

4 Results

After implementing the algorithm and running the code, we get results that are clearly not correct. This is due to indexing bugs from using a collocated grid and a staggered grid, as well as using the incorrect physical and temporal resolutions. However, after lots of troubleshooting, we finally get a result that makes sense. The figure below is screenshots of the final result at different times. When compared to experimental results of flow over a cylinder [3], there is a lot of similarity. Even more, the complex phenomenon known as vortex shedding is captured in this algorithm. Vortex shedding is a phenomenon that occurs when a fluid flows past a bluff body, such as a cylinder or a sphere, resulting in the formation of alternating vortices in the wake of the body.

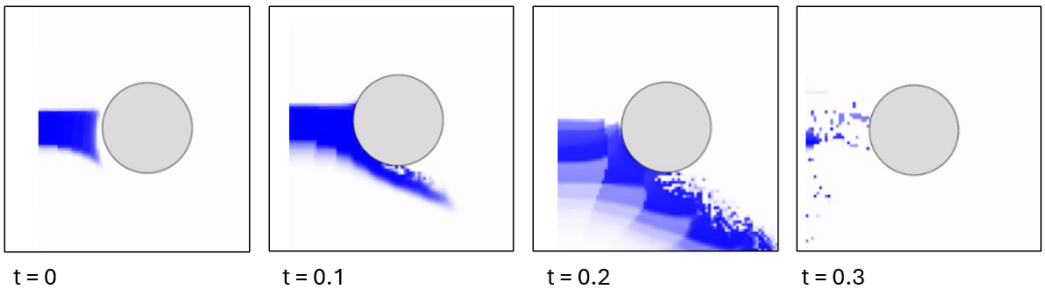


Figure 12: Screenshots of a Failed Test

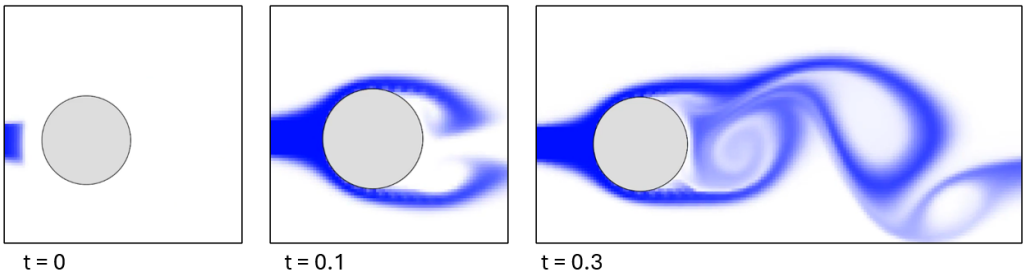


Figure 13: Screenshots of a Successful Test

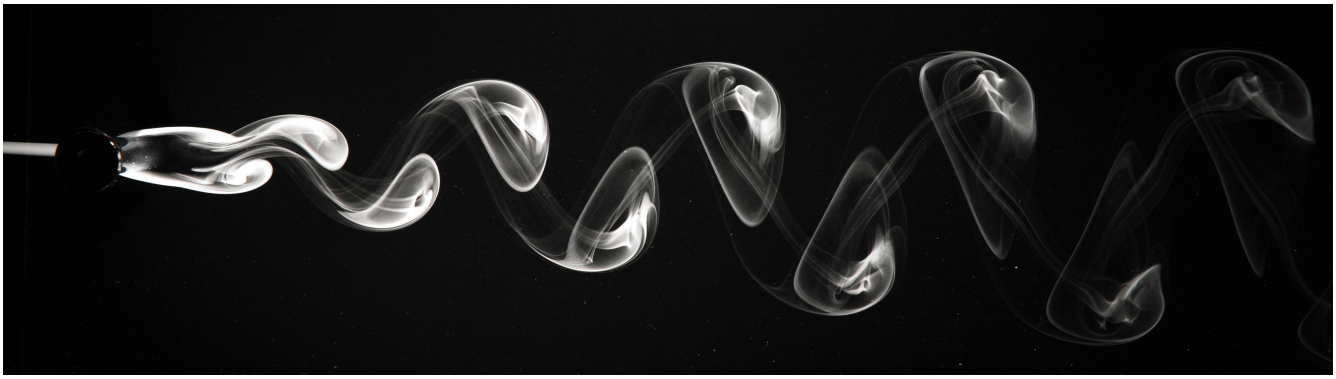


Figure 14: Experimental result of flow over a cylinder

Although the simulation was able to capture vortex shedding, it did not accurately represent the fluid boundary layer. The boundary layer is a thin layer of fluid adjacent to a solid surface where shear forces and viscous effects are large and consequently not negligible. However, since the boundary layer is extremely thin it does not affect the accuracy of the rest of the simulation.

Lastly, the simulation was fast. Compared to typical CFD software that can take days, weeks, or even months, this FFD simulation took mere seconds to run. Even more, further optimization of code could lead this simulation run in real-time.

5 Conclusion

In conclusion, this FFD algorithm has the necessary accuracy and speed to be more useful than standard CFD software in applications where speed is required . Furthermore, it is possible to extend this algorithm to more complex fluid flows with the addition of a few steps. More specifically, further work can be done to account for viscous and compressibility effects, as well extend the domain to three dimensions. It is also simple to apply this algorithm to other shapes such as airfoils. For these reasons, we recommend the use of this FFD algorithm when fast fluid simulations are desired.

References

[1] Edward J. Shaughnessy. Introduction to fluid mechanics / edward j. shaughnessy, ira m. katz, james p. schaffer., 2005.

[2] Jos Stam. Stable fluids - fast fluid dynamics algorithm. *ACM SIGGRAPH 99*, 1999, 11 2001.

[3] Jurgen Wagner. Karmansche wirbelstr kleine, 2014.

[4] Jun Zhang. Fast and high accuracy multigrid solution of the three dimensional poisson equation. *Journal of Computational Physics*, 143(2):449–461, 1998.

[5] Shijun Zhang. Notes from mathematics 361 - numerical analysis, linear systems sensitivity and iterative methods, 2024.