WebTunnel: A Zero-Dependency Eulerian Fluid Solver

Enrique Garcia Rivera

Abstract

This report details the mathematical and algorithmic implementation of **WebTunnel**, a lightweight Computational Fluid Dynamics (CFD) solver running entirely in client-side JavaScript. The simulation utilizes a grid-based Eulerian approach to solve the incompressible Navier-Stokes equations. By employing a semi-Lagrangian advection scheme and an implicit diffusion solver (Gauss-Seidel relaxation), the application achieves real-time performance and unconditional stability on standard web browsers without the need for WebGL or external libraries.

1 Mathematical Model

The simulation solves the incompressible Navier-Stokes equations, which describe the motion of fluid substances. For a velocity field **u** and scalar field (dye density) d, the equations are:

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

$$\nabla \cdot \mathbf{u} = 0 \tag{2}$$

Where:

- $-(\mathbf{u} \cdot \nabla)\mathbf{u}$ represents **Advection** (velocity moving itself).
- $-\nabla p$ represents **Pressure** gradients.
- $\nu \nabla^2 \mathbf{u}$ represents **Diffusion** (viscosity).
- $\mathbf{f}$ represents external forces (mouse interaction/wind inflow).
- $\nabla \cdot \mathbf{u} = 0$ represents the **Incompressibility** constraint (mass conservation).

2 Numerical Implementation

The solver is implemented on a uniform 2D grid. To maintain high performance in a single-threaded JavaScript environment, the 2D grid is flattened into 1D Float32Arrays. The solver relies on the method of *Operator Splitting*, where the terms of the Navier-Stokes equation are solved sequentially in discrete time steps (Δt).

2.1 1. Advection (Semi-Lagrangian Scheme)

Directly solving the advection term is computationally expensive and often unstable. WebTunnel employs a **Semi-Lagrangian** backtracing method. For every cell center **x** at time $t + \Delta t$, we trace the velocity field backwards to find the particle's position at time t:

$$\mathbf{x}_{old} = \mathbf{x} - \mathbf{u}(\mathbf{x})\Delta t$$

We then interpolate the value at $\mathbf{x}_{old}$ and assign it to the new cell. This method is unconditionally stable, allowing for larger time steps than explicit Eulerian schemes.

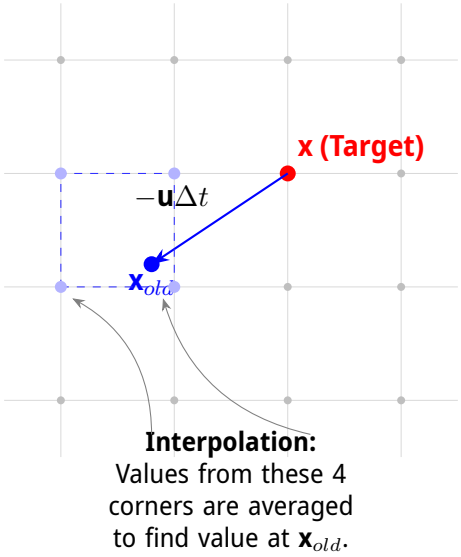

Figure 1: Visual representation of the Semi-Lagrangian backtracing step. To find the new value at **x**, the solver traces the velocity field backwards to $\mathbf{x}_{old}$ and interpolates the value from the surrounding grid points.

2.2 2. Diffusion (Implicit Integration)

Viscosity is solved using an implicit method to ensure stability when viscosity is high. This results in a system of linear equations:

$$\mathbf{u}^{new} = \mathbf{u}^{old} + \nu \Delta t \nabla^2 \mathbf{u}^{new}$$

This sparse linear system is solved using **Gauss-Seidel relaxation**, an iterative method that converges quickly for this specific matrix structure.

2.3 3. Projection (Mass Conservation)

After advection and diffusion, the velocity field is likely "divergent" (fluid is compressing or expanding). To enforce incompressibility ($\nabla \cdot \mathbf{u} = 0$), we apply the **Helmholtz-Hodge Decomposition**. We calculate the divergence of the field, solve a Poisson equation for pressure (p), and subtract the gradient of pressure from the velocity field:

$$\mathbf{u}_{divergence-free} = \mathbf{u} - \nabla p$$

This step is responsible for the curling vortices seen in the wake of obstacles.

3 Feature: Arbitrary Boundary Conditions

WebTunnel includes a rasterization pipeline that converts binary image data into boundary constraints.

- **Input:** User uploads a PNG/JPG image.
- **Rasterization:** Dark pixels are mapped to an 'obstacle' boolean grid.
- **Boundary Enforcement:** During the linear solve (Gauss-Seidel), velocities inside obstacle cells are forced to zero (Dirichlet condition), effectively creating a no-slip boundary condition around complex geometries like airfoils or text.

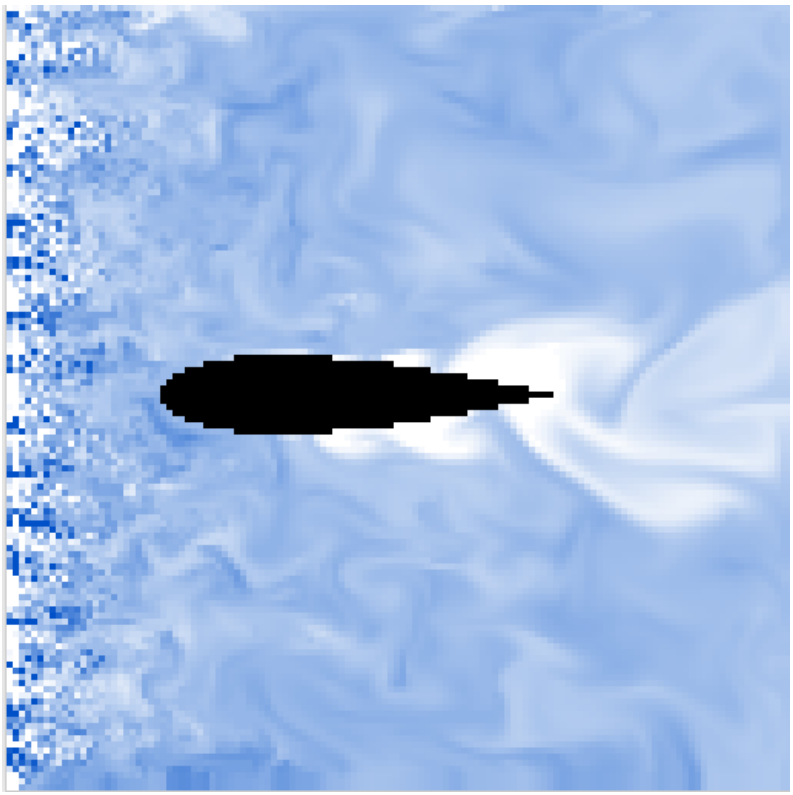

Figure 2: Simulation output showing vortex shedding behind a NACA 0020 airfoil.

4 Performance Optimization

To ensure 60 FPS performance on standard hardware:

1. **Typed Arrays:** `Float32Array` is used for all physics calculations to minimize memory overhead.
2. **Pointer Arithmetic Simulation:** A macro function `IX(x, y)` maps 2D coordinates to 1D indices to improve cache locality compared to nested arrays (e.g., `arr[x][y]`).
3. **Visuals:** Rendering is done via direct pixel manipulation of the HTML5 Canvas `ImageData` buffer, bypassing the slower vector drawing API.

5 Conclusion

WebTunnel demonstrates that complex physical simulations can be democratized via the web platform. By implementing Jos Stam's stable fluid solver in vanilla JavaScript, we achieve a portable, interactive wind tunnel useful for educational visualization and rapid aerodynamic prototyping.