Mathematical Modeling and of 2D Incompressible Fluid Flow

Ş. K. Alkır

The following document presents a detailed and formal derivation of the equations underlying the fluid simulation above. It explains how the mathematical model is constructed from physical laws—mass conservation, momentum balance, and incompressibility—and how these are translated into the numerical scheme. The exposition emphasizes both conceptual clarity and mathematical precision, making it suitable for readers seeking to understand the theoretical principles behind the simulation and how they guide its implementation.

1 Governing equations

Let $\Omega = [-1, 1]^2 \subset \mathbb{R}^2$ be a unit square with Cartesian coordinates $\boldsymbol{x} = (x, y)$, and let $t \geq 0$ denote time. The unknown fields are

$$\rho(\boldsymbol{x}, t) \in \mathbb{R}_{\geq 0}, \qquad \boldsymbol{u}(\boldsymbol{x}, t) = (u, v)^\top \in \mathbb{R}^2, \qquad p(\boldsymbol{x}, t) \in \mathbb{R}.$$

They satisfy the incompressible Navier–Stokes system with a passive-scalar density:

$$\partial_t \rho + \boldsymbol{u} \cdot \boldsymbol{\nabla} \rho = D_\rho \, \Delta \rho + S_\rho, \tag{1a}$$

$$\partial_t \boldsymbol{u} + (\boldsymbol{u} \cdot \boldsymbol{\nabla}) \boldsymbol{u} = -\boldsymbol{\nabla} p + \nu \, \Delta \boldsymbol{u} + \boldsymbol{f}, \tag{1b}$$

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0. \tag{1c}$$

1.1 Physical interpretation and modelling rationale

(i) Passive–scalar transport. Treat (1a) as a *control–volume budget*. The left–hand side is the *material derivative* of the concentration: it measures the instantaneous inventory change of a moving fluid parcel. On the right, $D_\rho \nabla^2 \rho$ is the divergence of the diffusive flux (Fick's law), spreading the scalar down its own gradient, while S_ρ is an external source representing user injections. In short, **material change = diffusive import + prescribed input**— a microscopic statement of the first law for a passive tracer.

(ii) Momentum balance (Navier–Stokes). The equation (1b) is Newton's second law written *per unit mass*. The material acceleration on the left is the "expense column" in our ledger; it quantifies how rapidly a fluid element's velocity state is being updated. The right-hand "revenue column" itemises the forces that can pay that expense: $-\nabla p$ is the reversible work done by pressure, $\nu \nabla^2 \boldsymbol{u}$ is the irreversible viscous diffusion of momentum, and $\boldsymbol{f}$ is an externally imposed specific force (our programmable wind). The equality therefore asserts a pointwise force balance: **material acceleration = pressure drive + viscous drag + external forcing**.

(iii) Incompressibility constraint. Finally, (1c) is the local volume-conservation ledger. The divergence operator tallies "flow-in minus flow-out" for an infinitesimal cube; setting that tally to zero demands that no fluid mass is created or destroyed and that density remains constant along trajectories. This requirement motivates the projection step used later (Section 6) and underlies the ubiquitous "pressure–Poisson" formulation in fluid solvers. Thus, the statement is a strict bookkeeping rule: **incoming volumetric flux − outgoing volumetric flux = 0**, everywhere and always, anchoring the solver to the physics of incompressible flow.

Why projection? Equation (1c) imposes a *pointwise* divergence-free constraint. A direct time-stepping of (1b) would not preserve $\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0$ at the discrete level. Chorin's projection method resolves this by splitting the update into a provisional step followed by an orthogonal projection onto the space of solenoidal vector fields (Section 6).

2 Spatial discretisation

2.1 Centered grid & indexing

The domain is tessellated by a uniform Cartesian grid with N interior cells per direction; in code $N = 200$. For simplicity we store *all* variables at cell centres (`stable fluids` layout) and add a *one-cell halo* to encode boundary conditions. The linear index

$$\mathrm{IX}(i, j) \;=\; i + (N + 2)\, j, \qquad 0 \le i, j \le N + 1,$$

maps the 2-D lattice to the 1-D `Float32Array` buffers (`u`, `v`, ...). With $(\Delta x, \Delta y) = (h, h)$ and $h = 1/N$, grid points are located at $(x_i, y_j) = ((i - \frac{1}{2})h, (j - \frac{1}{2})h)$ for $1 \le i, j \le N$.

3 Temporal discretisation

A fixed Eulerian time step $\Delta t = \mathtt{dt} = 0.01$ is used. The algorithm is operator-split into the following sub-steps (cf. Algorithm 1):

1. **Velocity forcing** by the wind field $\boldsymbol{f}$ (explicit Euler).
2. **Diffusion** of $\boldsymbol{u}$ via an *implicit* (unconditionally stable) solve §4.
3. **Projection** to enforce $\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0$ (§6).
4. **Advection** of $\boldsymbol{u}$ using semi-Lagrangian backtracing (§5).
5. **Second projection** (Strang-type splitting).
6. **Transport** of ρ: diffusion then advection.
7. **Artificial damping** (velocity, density, boundary).

Each step is mathematically well defined and conservative with respect to the quantities it advances; the only explicit dissipations are the physical ν, D_ρ and the user-adjustable damping factors $\gamma_v < 1, \gamma_\rho < 1$.

4 Diffusion step

4.1 Implicit time integration and unconditional stability

For a generic scalar field $\phi \in \{u, v, \rho\}$, representing either a velocity component or the passive density, the physical model reads

$$\partial_t \phi \;=\; \alpha\, \Delta \phi, \qquad \alpha > 0. \tag{2}$$

Let $t^n = n\Delta t$ and denote by $\phi^n_{i,j} \approx \phi(x_i, y_j, t^n)$ the grid solution at $(i, j) \in \{1, \dots, N\}^2$. Backward Euler in time combined with the five-point discrete Laplacian gives

$$\phi^{n+1}_{i,j} - \phi^n_{i,j} \;=\; a\Big(\phi^{n+1}_{i-1,j} + \phi^{n+1}_{i+1,j} + \phi^{n+1}_{i,j-1} + \phi^{n+1}_{i,j+1} - 4\phi^{n+1}_{i,j}\Big), \qquad a := \frac{\alpha \Delta t}{h^2}. \tag{3}$$

Symmetric positive definiteness. Rewriting (3) in vector form $(I - aL)\boldsymbol{\phi}^{n+1} = \boldsymbol{\phi}^n$, the matrix $A := I - aL$ is an M-matrix: symmetric, strictly diagonally dominant, and therefore positive definite for all $a > 0$. This guarantees uniqueness of $\boldsymbol{\phi}^{n+1}$ and makes the Jacobi (point) iteration converge geometrically. Moreover, the scheme is *unconditionally L^2-stable*. Indeed, multiplying (3) by $\phi_{i,j}^{n+1}$ and summing over all grid indices yields the discrete energy decay

$$\|\phi^{n+1}\|_2^2 - \|\phi^n\|_2^2 \;=\; -\,a\,\|\nabla_h \phi^{n+1}\|_2^2 \;\le\; 0,$$

so no CFL restriction on Δt is required.

Boundary operators. The helper `setBoundary` enforces:

$$\begin{aligned} \phi_{0,j} &= \pm\,\phi_{1,j}, \quad \phi_{N+1,j} = \pm\,\phi_{N,j}, \\ \phi_{i,0} &= \pm\,\phi_{i,1}, \quad \phi_{i,N+1} = \pm\,\phi_{i,N}, \end{aligned} \qquad 1 \le i, j \le N,$$

with the sign choice + (Neumann) for scalar fields and the sign − (Dirichlet) for the normal velocity components; this choice preserves the discrete energy estimate used in the stability argument.

4.2 Consistency and global error

The truncation error of (3) is $\mathcal{O}(\Delta t + h^2)$, so the scheme is first-order accurate in time and second-order in space. Combined with the L^2-stability above, the Lax–Richtmyer theorem yields global error $\mathcal{O}(\Delta t + h^2)$ for smooth solutions.

5 Advection step

5.1 Geometric interpretation and continuous formulation

Let $\boldsymbol{w} : \Omega \times [0, T] \to \mathbb{R}^d$ denote a scalar or vector field being transported by a time-dependent velocity $\boldsymbol{u}(\boldsymbol{x}, t)$. The advection equation

$$\partial_t \boldsymbol{w} + \boldsymbol{u} \cdot \nabla \boldsymbol{w} = 0 \qquad \textit{with} \quad \boldsymbol{w}(\boldsymbol{x}, t_0) = \boldsymbol{w}^0(\boldsymbol{x}) \tag{4}$$

This means that as each fluid parcel moves through space, it carries its initial field value with it unchanged. To describe this transport formally, we introduce the *flow map* $\boldsymbol{X}(t_0; t, \boldsymbol{x})$, which gives the position at time t_0 of the fluid particle that arrives at location $\boldsymbol{x}$ at time t.

Then the solution to (4) at time $t^{n+1} = t^n + \Delta t$ is:

$$\boldsymbol{w}^{n+1}(\boldsymbol{x}) = \boldsymbol{w}^n\big(\boldsymbol{X}(t^n; t^{n+1}, \boldsymbol{x})\big), \tag{5}$$

This is conceptually natural: to find the field value at point $\boldsymbol{x}$ and time t^{n+1}, trace the characteristic backwards to where the fluid parcel came from, and recover its earlier value there.

5.2 Semi-Lagrangian discretisation

Computing the departure point $\boldsymbol{X}(t^n; t^{n+1}, \boldsymbol{x})$ exactly would mean following the particle's path through the velocity field for an entire time step—a costly operation in real time. Instead, we approximate the backward trajectory by a single explicit Euler step:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \Delta t\, \boldsymbol{u}^n(\boldsymbol{x}).$$

Because $\tilde{\boldsymbol{x}}$ is almost never located *exactly* on a grid node, we must estimate the previous field value $\boldsymbol{w}^n$ there. Define the interpolation operator $\mathcal{I} : \{\boldsymbol{w}_{i,j}\} \to C^0(\Omega)$ by

$$\mathcal{I}[\boldsymbol{w}^n](x,y) \;:=\; \sum_{m=0}^{1}\sum_{n=0}^{1} \lambda_{mn}(x,y)\, \boldsymbol{w}^n_{i+m,\,j+n}\;,$$

where (i,j) is the lower-left grid node of the cell that contains (x,y) and the bilinear shape functions are

$$\lambda_{00} = (1-\xi)(1-\eta), \quad \lambda_{10} = \xi(1-\eta), \quad \lambda_{01} = (1-\xi)\eta, \quad \lambda_{11} = \xi\eta,$$

with $\xi = \dfrac{x - x_i}{h} \in [0,1]$, $\eta = \dfrac{y - y_j}{h} \in [0,1]$. Each $\lambda_{mn} \geq 0$ and $\sum_{m,n} \lambda_{mn} = 1$, so the operator is *contractive* in every L^p norm.

The semi-Lagrangian update therefore reads

$$\boldsymbol{w}^{n+1}(\boldsymbol{x}) \;=\; \mathcal{I}[\boldsymbol{w}^n](\tilde{\boldsymbol{x}}), \tag{6}$$

i.e. "sample the old field at the estimated upstream location." This construction mimics a *Lagrangian* perspective—track what a fluid particle carries—while keeping the data fixed on an Eulerian grid.

The key advantage is unconditional stability: regardless of Δt, the method cannot introduce new extrema, since it simply samples a convex combination of previous values.

Stability. Because update (6) merely interpolates existing data, it is contractive in every L^p norm, $\|\boldsymbol{w}^{n+1}\|_p \leq \|\boldsymbol{w}^n\|_p$ for $1 \leq p \leq \infty$; hence no Courant restriction and no spurious extrema arise. This unconditional stability—at first-order accuracy in time and second-order in space—is the key trade-off that makes the method ideal for real-time graphics.

6 Pressure projection

At each time step, the provisional velocity field $\boldsymbol{u}^*$ produced by forcing, diffusion, or advection may violate the incompressibility constraint $\nabla \cdot \boldsymbol{u} = 0$. To restore divergence-freeness, we perform a *projection* onto the subspace of solenoidal fields.

Helmholtz decomposition and pressure correction

Let $\boldsymbol{v} : \Omega_h \to \mathbb{R}^2$ be a discrete vector field defined on a regular Cartesian grid Ω_h with uniform spacing $h > 0$. A fundamental result in vector calculus, known as the *Helmholtz decomposition*, states that any sufficiently regular vector field can be uniquely decomposed as the sum of a divergence-free component and a gradient field. On the discrete grid, this takes the form

$$\boldsymbol{v} = \boldsymbol{v}_{\text{div}} + \nabla_h \pi, \tag{7}$$

where:

- $\boldsymbol{v}_{\text{div}}$ is a vector field satisfying $\nabla_h^* \boldsymbol{v}_{\text{div}} = 0$, i.e., it is discretely divergence-free,
- $\nabla_h \pi$ is the discrete gradient of a scalar field $\pi : \Omega_h \to \mathbb{R}$,
- ∇_h^* denotes the discrete divergence operator, defined as the negative adjoint of the gradient with respect to the grid inner product.

This decomposition is orthogonal in the discrete L^2 inner product and is unique up to an additive constant in π.

On the regular grid with spacing h, the discrete gradient operator ∇_h maps scalar grid functions $\pi_{i,j}$ to vector fields via central differences:

$$(\nabla_h \pi)_{i,j} = \left(\frac{\pi_{i+1,j} - \pi_{i-1,j}}{2h}, \frac{\pi_{i,j+1} - \pi_{i,j-1}}{2h} \right).$$

The discrete divergence operator ∇_h^* acts on a vector field $\boldsymbol{v}_{i,j} = (u_{i,j}, v_{i,j})$ as:

$$(\nabla_h^* \boldsymbol{v})_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2h} + \frac{v_{i,j+1} - v_{i,j-1}}{2h}.$$

These satisfy the identity $\langle \boldsymbol{v}, \nabla_h \pi \rangle_h = -\langle \nabla_h^* \boldsymbol{v}, \pi \rangle_h$ under the grid L^2 inner product, confirming that ∇_h^* is indeed the negative adjoint of ∇_h.

In the context of incompressible fluid simulation, we apply this decomposition to the provisional velocity field $\boldsymbol{u}^*$, which results from the combination of advection, external forcing, and viscosity but does not generally satisfy the incompressibility condition $\nabla \cdot \boldsymbol{u} = 0$.

To recover a physically admissible velocity field $\boldsymbol{u}^{n+1}$ that is divergence-free, we define:

$$\boldsymbol{u}^{n+1} := \boldsymbol{u}^* - \nabla_h \pi,$$

where π is chosen such that the divergence of $\boldsymbol{u}^{n+1}$ vanishes:

$$\nabla_h^* \boldsymbol{u}^{n+1} = \nabla_h^* (\boldsymbol{u}^* - \nabla_h \pi) = \nabla_h^* \boldsymbol{u}^* - \Delta_h \pi = 0.$$

This leads to the discrete Poisson equation for the pressure correction:

$$\Delta_h \pi = \nabla_h^* \boldsymbol{u}^*, \tag{8}$$

where $\Delta_h := \nabla_h^* \nabla_h$. In practice, the Poisson equation (8) is solved numerically using the same iterative Jacobi (point) method as in the diffusion step; see Section 4.

We impose homogeneous Neumann boundary conditions for π, corresponding to the physical condition of no normal flow at the domain boundaries:

$$\partial_n \pi = 0 \quad \text{on } \partial\Omega_h.$$

Since π is determined only up to an additive constant, the solution to (8) is not unique. However, the velocity field $\boldsymbol{u}^{n+1}$ is fully determined, because the gradient of a constant vanishes.

In summary, the pressure correction π serves to remove the non-solenoidal component of the provisional velocity field $\boldsymbol{u}^*$, producing an updated velocity $\boldsymbol{u}^{n+1}$ that satisfies the incompressibility condition exactly at the discrete level.

This projection step is applied both before and after the advection stage as part of a Strang-type operator splitting; see Section 9 for the full time-stepping sequence.

Stability and energy estimate

By orthogonality of the decomposition, the projection does not increase kinetic energy:

$$\|\boldsymbol{u}^{n+1}\|_2 = \|\boldsymbol{u}^*\|_2 - \|\nabla_h \pi\|_2,$$

ensuring that no artificial energy is introduced when enforcing incompressibility. This mirrors the continuous L^2-energy estimate and preserves the dissipative nature of the Navier–Stokes evolution.

The discrete L^2 norm is defined as

$$\|\boldsymbol{u}\|_2 := h^2 \sum_{i,j} \left(u_{i,j}^2 + v_{i,j}^2 \right),$$

with the corresponding inner product $\langle \boldsymbol{u}, \boldsymbol{v} \rangle := h^2 \sum_{i,j} (u_{i,j} v_{i,j} + v_{i,j} w_{i,j})$.

7 Wind forcing

The external force field $\boldsymbol{f}(x, y)$, referred to as the wind field, is defined pointwise on the domain according to one of the following user-selectable modes:

$$\boldsymbol{f}(x,y) = \begin{cases} f_0 \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}, & \text{Uniform (constant direction)}, \\ f_0 \begin{pmatrix} -\widehat{y}(x,y) \\ \widehat{x}(x,y) \end{pmatrix}, & \text{Vortex (solid-body rotation)}, \\ f_0 \begin{pmatrix} \sin(\pi x) \\ \cos(\pi y) \end{pmatrix}, & \text{Sinusoidal}, \\ f_0 \begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}, & \text{Custom (user-defined)}, \end{cases}$$

where:

- $f_0 \in [0, 20]$ is the user-defined magnitude of the wind,
- $\theta \in [0, 2\pi)$ specifies the angle of uniform flow,
- $(\widehat{x}(x,y), \widehat{y}(x,y)) := \frac{(x-x_c,\, y-y_c)}{\sqrt{(x-x_c)^2+(y-y_c)^2}}$ is the unit radial vector from the domain center (x_c, y_c) (used in the vortex mode),
- $f(x, y)$ and $g(x, y)$ are arbitrary scalar functions supplied by the user in the Custom mode.

Note that all functions are defined over the rescaled domain $[-1, 1]^2$, where the origin corresponds to the center of the canvas.

8 Density injection and damping

User interaction is implemented as an external source term in the density equation (1a). Specifically, a mouse drag triggers the addition of a source term

$$S_\rho(x, y, t) = \rho_0 \, \mathbf{1}_{B_r(x_0,y_0)}(x, y),$$

where ρ_0 is the prescribed injection strength, and $B_r(x_0, y_0)$ is a ball of radius r centered at the injection location (x_0, y_0). The indicator function $\mathbf{1}_{B_r}$ ensures that the density is injected only within this local region. In addition to modifying ρ, a small velocity impulse is applied to $\boldsymbol{u}$ in the direction of the drag, simulating physical stirring.

To prevent unbounded accumulation of density and numerical artefacts during long-time integration, artificial damping is applied after each time step:

$$\boldsymbol{u} \leftarrow \gamma_v \, \boldsymbol{u}, \qquad \rho \leftarrow \gamma_\rho \, \rho, \qquad \text{with} \quad \gamma_v, \gamma_\rho \lesssim 1.$$

These coefficients serve as phenomenological models for subgrid-scale dissipation and help stabilize the simulation while preserving large-scale dynamics. Both γ_v and γ_ρ are user-configurable parameters.

9 Algorithmic summary

Algorithm 1 Advance one time step $t^n \to t^{n+1}$

procedure STEP
  if mouse is dragging **then**
    Inject: $\rho \mathrel{+}= S_\rho,\ u \mathrel{+}= \Delta u,\ v \mathrel{+}= \Delta v$ ▷ density source and stirring impulse
  end if
  Force: $u[k] \mathrel{+}= \Delta t\,\mathrm{windU}[k]$, $v[k] \mathrel{+}= \Delta t\,\mathrm{windV}[k]$ ▷ apply wind field
  Diffuse: DIFFUSE(1, u_0, u, ν); DIFFUSE(2, v_0, v, ν) ▷ implicit viscous solve
  Project: PROJECT($u_0, v_0, p, \mathrm{div}$) ▷ enforce $\nabla \cdot \boldsymbol{u} = 0$
  Advect: ADVECT(1, u, u_0, u_0, v_0); ADVECT(2, v, v_0, u_0, v_0) ▷ semi-Lagrangian transport
  Project: PROJECT(u, v, p, div) ▷ remove post-advection divergence
  Density steps: DIFFUSE(0, ρ_0, ρ, D_ρ); ADVECT(0, ρ, ρ_0, u, v) ▷ diffuse then advect scalar
  Damping: $u \leftarrow \gamma_v\,u$, $v \leftarrow \gamma_v\,v$, $\rho \leftarrow \gamma_\rho\,\rho$ ▷ global decay
  Edge damping: halve ρ on boundary cells ▷ suppress artefacts at domain edges
  Render: RENDERDENSITY;
  if `showField` **then** DRAWFIELD; DRAWAXES
  end if
end procedure

Further reading.

- J. Stam, *Stable Fluids*, SIGGRAPH 1999.
- E. Guendelman et al., *Semi-Lagrangian Controllable Fluids*, ACM TOG 2012.
- R. Bridson, *Fluid Simulation for Computer Graphics*, 3rd ed., CRC Press, 2015. A concise, code-oriented textbook that covers projection solvers, semi-Lagrangian advection, vorticity confinement, and GPU tips.
- J. Kim and P. Moin, "Application of a fractional-step method to incompressible Navier–Stokes equations," *J. Comp. Phys.*, 59 (1985) 308-323. The classic finite-difference projection paper used by many CFD codes.