Frame Update Thoroughly

To enforce incompressibility and remove the changes in pressure, we attempt to zero out the divergence throughout the flow field. This is done by adding flow away from high pressure converging areas, and toward low pressure diverging areas. A gradient is the slope or rate of a value across a grid. The gradient of the divergence is a vector field that points from high pressure to low pressure areas. So to remove the divergence from a flow field, we repeatedly increment it by the gradient of its divergence.

We can combine the divergence and gradient calculations, and if we find divergence values at the corners between grid cells instead of at their centers, we can use a simple 3x3 convolution for this.

The Navier Stokes Equations

The Navier-Stokes equations for incompressible constant density fluids:

$$\frac{\partial \vec{u}}{\partial t} = -\frac{\nabla p}{\rho} - (\vec{u} \cdot \nabla)\vec{u} + \nu \nabla^2 \vec{u} + \vec{f},$$
$$\nabla \cdot \vec{u} = 0.$$

Discretized Fields

We have discretized the grid into $N_y \times N_x$ cells such that each cell has their own (implicit) discrete $(p)_{i,j}$ values in the center and each cell has velocity field values $(u_y)_{i\pm\frac{1}{2},j}$ and $(u_x)_{i,j\pm\frac{1}{2}}$ on the corresponding edges that correspond to the velocity components entering and exiting the cell. Therefore the dimensions of each of these matrices will be:

$$(p)_{i,j} \in \mathbb{R}^{N_y \times N_x},$$
$$(u_x)_{i\pm\frac{1}{2},j} \in \mathbb{R}^{(N_y+1)\times N_x},$$
$$(u_y)_{i,j\pm\frac{1}{2}} \in \mathbb{R}^{N_y \times (N_x+1)}.$$

(The half indices are used just to with respect to the cell indices). Each cell a has spatial dimension $\Delta x \times \Delta y$ where $\Delta x = \Delta y = h$, i.e it is the smallest spatial ("approximated infinitesimal") spacing parameter that will be used in measuring the domain and calculating spatial finite differences. Minimal time evolution spacing i.e the time between each physical frame or tick will be Δt.

```
// --- Parameters ---
Nx, Ny        // number of cells in x and y
dx, dy        // cell size
dt            // timestep
rho           // fluid density
```

```
// --- Define Arrays ---

// pressure: cell-centered
p[0..Nx-1, 0..Ny-1]

// scalar fields: cell-centered
dye[0..Ny-1, 0..Nx-1]
temperature[0..Nx-1, 0..Ny-1]

u_x[0..Ny-1, 0..Nx]      // x-velocity on horizontal edges
u_y[0..Ny, 0..Nx-1]      // y-velocity on vertical edges

// divergence: cell-centered
div[0..Nx-1, 0..Ny-1]

// boundary mask
isSolid[0..Nx-1, 0..Ny-1]  // 1 if solid, 0 if fluid

// --- Initialization ---
// zero all velocities
for i = 0 to Nx:
    for j = 0 to Ny-1:
        u_x[i][j] = 0

for i = 0 to Nx-1:
    for j = 0 to Ny:
        u_y[i][j] = 0

// zero pressure
for i = 0 to Nx-1:
    for j = 0 to Ny-1:
        p[i][j] = 0
        div[i][j] = 0
        dye[i][j] = 0
        temperature[i][j] = 0
        isSolid[i][j] = 0   // default: fluid

// optionally already set solid boundaries around the domain
for i = 0 to Nx-1:
    isSolid[i][0] = 1      // bottom wall
    isSolid[i][Ny-1] = 1   // top wall

for j = 0 to Ny-1:
    isSolid[0][j] = 1      // left wall
    isSolid[Nx-1][j] = 1   // right wall
```

Boundary Masks

To make the simulation more interesting, boundaries will be added within the simulation domain. Let $(D)_{i,j} \in \mathbb{R}^{N_y \times N_x}$ be the spatial domain matrix of the simulation where each coordinate corresponds to the i, j-th cell. In programming, it could just be a bit-matrix where one-values indicate, that this cell is a part of the fluid, and zero-values show, that the cell is a part of some obstacle.
Therefore within the update loop, the following check will be applied:

$$\text{if } (D)_{i,j} = 0 \text{ then } (u_x)_{i,j\pm\frac{1}{2}} = (u_y)_{i\pm\frac{1}{2},j} = 0.$$

Frame Update

Let k be the timestep, such that the current time of the simulation is $t_k = k\Delta t$ and the next step will be $t_{k+1} = (k+1)\Delta t$. Since some N operations will be applied on the velocity fields during one update loop, let $l = 1, 2, \ldots, N$ be the index refering to the current step such that the transformation of velocity fields will be the following:

$$\begin{aligned}
(u_x)^{[k]}_{i,j\pm\frac{1}{2}} \to (u_x)^{1}_{i,j\pm\frac{1}{2}} \to (u_x)^{2}_{i,j\pm\frac{1}{2}} \to \cdots \to (u_x)^{l}_{i,j\pm\frac{1}{2}} \to \cdots \to (u_x)^{N}_{i,j\pm\frac{1}{2}} = (u_x)^{[k+1]}_{i,j\pm\frac{1}{2}}, \\
(u_y)^{[k]}_{i\pm\frac{1}{2},j} \to (u_y)^{1}_{i\pm\frac{1}{2},j} \to (u_y)^{2}_{i\pm\frac{1}{2},j} \to \cdots \to (u_y)^{l}_{i\pm\frac{1}{2},j} \to \cdots \to (u_y)^{N}_{i\pm\frac{1}{2},j} = (u_y)^{[k+1]}_{i\pm\frac{1}{2},j},
\end{aligned}$$

where each arrow between some l-th and $l+1$-st value corresponds to the specific update step.

Addition of Forces

$$\frac{\partial \vec{u}}{\partial t} = \vec{f}.$$

Firstly, the contribution of the forcing term $\vec{f}$, which acts as acceleration, will be added to the velocity fields $(u_x)_{i,j\pm\frac{1}{2}}, (u_y)_{i\pm\frac{1}{2},j}$. The forcing term could be some preprogrammed time-dependent field or could instead be dependent on the user-input - such as applying some acceleration brush or moving the boundaries, latter of which will be tricky to implement.
The velocity change contribution from force will be acceleration times the time-step.

Let the discrete form of the forcing term $\vec{f}$ be denoted as a such as acceleration usually is. Then the discrete matrices of the acceleration values will have the same shape as the corresponding velocity fields i.e

$$\begin{aligned}
(a_x)_{i,j\pm\frac{1}{2}} &\in \mathbb{R}^{N_y \times (N_x+1)}, \\
(a_y)_{i\pm\frac{1}{2},j} &\in \mathbb{R}^{(N_y+1) \times N_x}.
\end{aligned}$$

The first transformation will then be

$$\begin{aligned}
(u_x)^{1}_{i,j\pm\frac{1}{2}} &= (u_x)^{[k]}_{i,j\pm\frac{1}{2}} + (a_x)_{i,j\pm\frac{1}{2}}\Delta t, \\
(u_y)^{1}_{i\pm\frac{1}{2},j} &= (u_y)^{[k]}_{i\pm\frac{1}{2},j} + (a_y)_{i\pm\frac{1}{2},j}\Delta t.
\end{aligned}$$

To make the notation more compact, let's rewrite the last set of four equations as

$$(\mathbf{u})_{i,j}^{\text{next}} = (\mathbf{u})_{i,j}^{[k]} + (\mathbf{a})_{i,j}\Delta t,$$

so that from now on, the expressions in bold indicate the four staggered values on the edges of each i, j-th cell.

If the places of added acceleration do not coincide with the boundaries then right-after the force application, the zero-velocity boundary conditions will be applied right after the iteration step.

Advection

$$\frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{u}.$$

To advect velocity components for each i, j-th fluid cell's four edges, we sample a new value from $\mathbf{u}^1$ velocity field components from position $\mathbf{x}_{\text{edge}} - (\mathbf{u})_{\text{edge}}^1 \Delta t$ so we basically backtrack and look for what possible value each face's velocity corresponds to. The backtracking will most of the times correspond to some arbitrary location with non-integer indices $(m + \Delta m, n + \Delta n)$ where corresponding Δ-s are the fractional parts between $-\frac{1}{2}$ and $\frac{1}{2}$, so to sample from these indices, bilinear interpolation will be used.

Therefore for each edge's velocity

$$\mathbf{u}^{\text{next}}(\mathbf{x}) = \mathbf{u}^{\text{prev}}(\mathbf{x} - \mathbf{u}^{\text{prev}}(\mathbf{x})\Delta t).$$

Let position vectors on the grid be defined as for some arbitrary index with fractional part

$$(\mathbf{x})_{a,b} = \begin{pmatrix} y_a \\ x_b \end{pmatrix} = \begin{pmatrix} ah \\ bh \end{pmatrix},$$

where h is the spacing of the grid.

Some arbitrary new position $\mathbf{x}_{a',b'} = \mathbf{x}_{a,b} - \mathbf{u}^1(\mathbf{x_{a,b}})\Delta t$ is then

$$\begin{aligned}
\begin{pmatrix} y_{a'} \\ x_{b'} \end{pmatrix} &= \begin{pmatrix} y_a \\ y_b \end{pmatrix} - \Delta t \begin{pmatrix} (u_y)_{a,b} \\ (u_x)_{a,b} \end{pmatrix}, \\
\begin{pmatrix} a'h \\ b'h \end{pmatrix} &= \begin{pmatrix} ah \\ bh \end{pmatrix} - \Delta t \begin{pmatrix} (u_y)_{a,b} \\ (u_x)_{a,b} \end{pmatrix}, \\
\implies a' &= a - \frac{1}{h}\Delta t (u_y)_{a,b}, \\
b' &= b - \frac{1}{h}\Delta (u_x)_{a,b}.
\end{aligned}$$

Therefore each edge's velocity is advected in the following way:

$$\begin{aligned}
(u_x)_{i,j\pm\frac{1}{2}}^{\text{next}} &= (u_x)_{m+\Delta m, n+\Delta n}^{\text{prev}}, \\
(u_y)_{i\pm\frac{1}{2},j}^{\text{next}} &= (u_y)_{p+\Delta p, q+\Delta q}^{\text{prev}},
\end{aligned}$$

where the sampling indices are

$$
\begin{aligned}
m + \Delta m &= i - \frac{1}{h}\Delta t(u_y)_{i,j\pm\frac{1}{2}}, \\
n + \Delta n &= j - \frac{1}{h}\Delta t(u_x)_{i,j\pm\frac{1}{2}}, \\
p + \Delta p &= i - \frac{1}{h}\Delta t(u_y)_{i\pm\frac{1}{2},j}, \\
q + \Delta q &= j - \frac{1}{h}\Delta t(u_x)_{i\pm\frac{1}{2},j}.
\end{aligned}
$$

If any of these calculated sampling indices are out of bounds or coincide with boundary, they will be recalculated to be the furthest fluid cell from i, j-th cell towards the sampled indices.

Notice, how there are values of $(u_x)_{i\pm\frac{1}{2},j}$ and $(u_y)_{i,j\pm\frac{1}{2}}$ which aren't well defined to be some elements of any grid. For finding these values, bilinear interpolation will be used. Also, the sampling indices will most certainly land on some arbitrary place in a cell, which also calls for bilinear interpolation when sampling.

Diffusion

The diffusion part of the Navier-Stokes equation is

$$
\frac{\partial \vec{u}}{\partial t} = \nu \nabla^2 \vec{u}.
$$

For applying diffusion on velocity, a backward implicit difference scheme is used rather than explicit forward scheme (let L be discrete Laplace operator):

$$
\frac{\mathbf{u}^{\text{next}} - \mathbf{u}^{\text{prev}}}{\Delta t} = \nu \nabla^2 \mathbf{u}^{\text{next}}.
$$

When setting $\nu = 0$, i.e simulating non-viscous fluid, then this step can be skipped and computational costs can be reduced.

Pressure Poisson Solve & Apply Pressure Gradient

We are left with the next part of NS that applies the pressure gradient to the field:

$$
\begin{aligned}
\frac{\partial \vec{u}}{\partial t} &= -\frac{1}{\rho}\nabla p, \\
\nabla \cdot \vec{u} &= 0.
\end{aligned}
$$

Since, pressure isn't known, it must be algebraically solved for such that the zero-divergence condition is satisfied.

Currently the velocity field solution from the earlier step $\mathbf{u}^{\text{old}}$ most certainly is not divergence free so we want to apply the gradient such that the $\mathbf{u}^{\text{new}}$ is divergence free. The finite difference version of the first term is:

$$
\begin{aligned}
&\frac{\mathbf{u}^{\text{new}} - \mathbf{u}^{\text{old}}}{\Delta t} = -\frac{1}{\rho}\nabla p, \\
&\nabla \cdot \mathbf{u}^{\text{new}} = 0, \\
\implies &\mathbf{u}^{\text{new}} - \mathbf{u}^{\text{old}} = -\frac{\Delta t}{\rho}\nabla p, \Big| \text{ apply divergence } \nabla \cdot () \\
\implies &\nabla^2 p = \frac{\rho}{\Delta t}\nabla \cdot \mathbf{u}^{\text{old}}.
\end{aligned}
$$

The equation

$$
\nabla^2 p = \frac{\rho}{\Delta t}\nabla \cdot \mathbf{u}^{\text{old}}
$$

is the Poisson equation for the implicit pressure that can be solved for pressure with different iterative methods. Most common and the easiest method to implement is the **Jacobi Iteration** which in this case iterativaly sets each $(p)_{i,j}$ value accross the grid to be the sum of the average of its four neighbors minus the value of the RHS of the pressure Poisson. More advanced methods that converge faster are **Gauss-Seidel**, **Successive Over-Relaxation**, **Conjugate Gradient** and the best but also the most difficult to implement is **Multigrid**.

Jacobi Iteration

Due to the staggeredness of the velocity fields, the divergence of the velocity for each i, j-th cell is just the sum of the finite differences of velocity components on the edges of the cell:

$$
\nabla \cdot (u)_{i,j} = \frac{(u_x)_{i,j+\frac{1}{2}} - (u_x)_{i,j-\frac{1}{2}}}{h} + \frac{(u_y)_{i+\frac{1}{2},j} - (u_y)_{i-\frac{1}{2},j}}{h}.
$$

The finite difference form of the pressure Laplacian is

$$
\nabla^2 (p)_{i,j} = \frac{(p)_{i,j+1} + (p)_{i,j-1} + (p)_{i+1,j} + (p)_{i-1,j} - 4(p)_{i,j}}{h^2}.
$$

Therefore the finite difference form of the Poisson equation becomes

$$
\begin{aligned}
&\frac{(p)_{i,j+1} + (p)_{i,j-1} + (p)_{i+1,j} + (p)_{i-1,j} - 4(p)_{i,j}}{h^2} = \frac{\rho}{\Delta t}\nabla \cdot (u)^{\text{old}}_{i,j}, \\
&\nabla \cdot (u)^{\text{old}}_{i,j} = \frac{(u_x)^{\text{old}}_{i,j+\frac{1}{2}} - (u_x)^{\text{old}}_{i,j-\frac{1}{2}} + (u_y)^{\text{old}}_{i+\frac{1}{2},j} - (u_y)^{\text{old}}_{i-\frac{1}{2},j}}{h}.
\end{aligned}
$$

For iterative solvers the pressure field can be initialized with some heuristically informed guesses or just as fields of zeroes. Let r be the index of an iteration step. Each $(p)_{i,j}$ can be iteratively solved as the following:

$$
(p)^{r+1}_{i,j} = \frac{(p)^{r}_{i+1,j} + (p)^{r}_{i-1,j} + (p)^{r}_{i-1,j} + (p)^{r}_{i-1,j}}{4} - \frac{\rho h^2}{4\Delta t}\nabla \cdot (u)^{\text{old}}_{i,j}.
$$

To maintain the structural integrity, solid non-fluid parts of the grid have pressures that match the pressures just on the outside of the boundaries. Therefore Neumann boundary

conditions are implemented, which state that

$$\frac{\partial p}{\partial \vec{n}} = 0,$$
$$\frac{p_{\text{just outside}} - p_{\text{boundary}}}{h} = 0.$$

Therefore, for example, if neighboring cell position with indices $i, j+1$ is a solid boundary, the iteration expression becomes

$$(p)_{i,j}^{r+1} = \frac{(p)_{i-1,j}^{r} + (p)_{i-1,j}^{r} + (p)_{i-1,j}^{r}}{3} - \frac{\rho h^2}{3\Delta t}\nabla \cdot (u)_{i,j}^{\text{old}},$$
$$\nabla \cdot (u)_{i,j}^{\text{old}} = \frac{0 - (u_x)_{i,j-\frac{1}{2}}^{\text{old}} + (u_y)_{i+\frac{1}{2},j}^{\text{old}} - (u)_{i-\frac{1}{2},j}^{\text{old}}}{h},$$

so in general, the denominator multiplier corresponds to the number of neigbors. Velocity components on the edges of the boundaries neighboring the fluid grid must be always set to zero.

The gradient of the pressure field across each i, j-th cell's edge is then calculated using finite difference schemes across the edges of the cells.

$$\text{For } (i, j-1/2)\text{-th edge:}$$
$$\nabla_x (p)_{i,j-\frac{1}{2}} = \frac{p_{i,j} - p_{i,j-1}}{h},$$
$$\text{for } (i, j+1/2)\text{-th edge:}$$
$$\nabla_x (p)_{i,j+\frac{1}{2}} = \frac{p_{i,j+1} - p_{i,j}}{h}$$
$$\text{for } (i-1/2, j)\text{-th edge:}$$
$$\nabla_y (p)_{i-\frac{1}{2},j} = \frac{p_{i,j} - p_{i-1,j}}{h},$$
$$\text{for } (i, j+1/2)\text{-th edge:}$$
$$\nabla_y (p)_{i+\frac{1}{2},j} = \frac{p_{i+1,j} - p_{i,j}}{h}.$$

After some hopefully sufficient amount of Jacobi iterations the pressure solution $(p)_{i,j}^{r_{\text{final}}}$ will be reached. The new velocity components therefore are updated as follows:

$$\mathbf{u}^{\text{new}} = \mathbf{u}^{\text{old}} - \frac{\Delta t}{\rho}\nabla p,$$
$$(u_x)_{i,j\pm\frac{1}{2}}^{\text{new}} = (u_x)_{i,j\pm\frac{1}{2}}^{\text{old}} - \frac{\Delta t}{\rho}\nabla_x (p)_{i,j\pm\frac{1}{2}}^{r_{\text{final}}},$$
$$(u_y)_{i\pm\frac{1}{2},j}^{\text{new}} = (u_y)_{i\pm\frac{1}{2},j}^{\text{old}} - \frac{\Delta t}{\rho}\nabla_y (p)_{i\pm\frac{1}{2},j}^{r_{\text{final}}}.$$

If across the fluid cells, the updated velocity doesn't satisfy near-zero divergence criteria, the Jacobi iteration will be reinitiated with setting $\mathbf{u}^{\text{old}} \leftarrow \mathbf{u}^{\text{new}}$.

After all of the previous steps $\text{add force} \rightarrow \text{advect} \rightarrow (\text{diffuse}) \rightarrow \text{clear divergence}$ we can set new the velocity field components as $\mathbf{u}^{[k+1]} \leftarrow \mathbf{u}_{\mathbf{new}}$ and also use the new field to update visual fields or particles simulated in the fluid.

Advecting Dye Fields and Particles