A Multigrid Poisson Solver for Fluid Simulation

Junlin Wu

December, 2018

1 Project Overview

The main goals of this project are to introduce and implement a geometric multigrid solver, compare it to classical iterative methods, analysis its performance, and further apply it as the solver for fluid simulation preconditioners.

2 Model Setup

We are interested in solving the Poisson problem in a domain $\Omega \in \mathbb{R}^n$, with boundary $\partial\Omega$, which can be expressed as

$$\begin{cases} -a\nabla^2 u = f & \text{in } \Omega \\ u = g & \text{on } \partial\Omega \end{cases} \tag{1}$$

where a is a known constant, f and g are functions in domain Ω and $\nabla^2 = \nabla \cdot \nabla(\cdot)$ is the Laplacian operator. In this paper we discuss the Poisson problem under the two dimensional scheme. For simplicity, we set the domain Ω as the unit square $[0,1]\times[0,1]$, $a = 1$ and $g \equiv 0$.

2.1 Discretization

In order to numerically solve the problem, we partition the unit square domain into equally spacing grid, where h_x is the grid distance over the x-axis, h_y is the grid distance over the y-axis. We set $h = h_x = h_y = \frac{1}{N}$. The points on the grid could be written as $(x_i, y_j) = (ih_x, jh_y)$, where $0 \le i \le N, 0 \le j \le N$.

We discretize the problem in Equation (1) using third order Taylor approximation on the grid set up above, for $1 \le i \le N-1, 1 \le j \le N-1$ we have

$$\begin{aligned}
u(x_{i+1}, y_j) &= u(x_i, y_j) + hu_x(x_i, y_j) + \frac{h^2}{2!}u_{xx}(x_i, y_j) + \frac{h^3}{3!}u_{xxx}(x_i, y_j) + O(h^4) \\
u(x_{i-1}, y_j) &= u(x_i, y_j) - hu_x(x_i, y_j) + \frac{h^2}{2!}u_{xx}(x_i, y_j) - \frac{h^3}{3!}u_{xxx}(x_i, y_j) + O(h^4) \\
u(x_i, y_{j+1}) &= u(x_i, y_j) + hu_y(x_i, y_j) + \frac{h^2}{2!}u_{yy}(x_i, y_j) + \frac{h^3}{3!}u_{yyy}(x_i, y_j) + O(h^4) \\
u(x_i, y_{j-1}) &= u(x_i, y_j) - hu_y(x_i, y_j) + \frac{h^2}{2!}u_{yy}(x_i, y_j) - \frac{h^3}{3!}u_{yyy}(x_i, y_j) + O(h^4)
\end{aligned} \tag{2}$$

Summing the equations above, we have

$$\begin{aligned} u_{xx}(x_i, y_j) &= \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} + O(h^2) \\ u_{yy}(x_i, y_j) &= \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{h^2} + O(h^2) \end{aligned} \tag{3}$$

Let v_{ij} be the approximation of $u(x_i, x_j)$, and $f_{ij} = f(x_i, y_j)$, the Equation (1) could be approximated as

$$\begin{aligned} \frac{-v_{i-1,j}+2v_{ij}-v_{i+1,j}}{h_x^2} + \frac{-v_{i,j-1}+2v_{ij}-v_{i,j+1}}{h_y^2} &= f_{ij} \qquad \text{for } 1 \le i, j \le N-1 \\ v_{ij} &= 0 \qquad \text{for } i, j \in 0, N \end{aligned} \tag{4}$$

2.2 Linear System

At this point, we are able to summarize the discretization results and form it into a linear system. The matrix expression of Equation (4) could be written as $Av = f$, where

$$A = \begin{bmatrix} A_1 & -I_y & & & & \\ -I_y & A_2 & -I_y & & & \\ & -I_y & A_3 & -I_y & & \\ & & & ..... & & \\ & & & -I_y & A_{N-2} & -I_y \\ & & & & -I_y & A_{N-1} \end{bmatrix} \quad v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ ... \\ v_{N-2} \\ v_{N-1} \end{bmatrix} \quad f = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ ... \\ f_{N-2} \\ f_{N-1} \end{bmatrix}$$

$$A_i = \begin{bmatrix} \frac{1}{h_x^2} + \frac{1}{h_y^2} & -\frac{1}{h_x}^2 & & & & \\ -\frac{1}{h_x}^2 & \frac{1}{h_x^2} + \frac{1}{h_y^2} & -\frac{1}{h_x}^2 & & & \\ & -\frac{1}{h_x}^2 & \frac{1}{h_x^2} + \frac{1}{h_y^2} & -\frac{1}{h_x}^2 & & \\ & & ..... & ..... & & \\ & & & -\frac{1}{h_x}^2 & \frac{1}{h_x^2} + \frac{1}{h_y^2} & -\frac{1}{h_x}^2 \\ & & & & -\frac{1}{h_x}^2 & \frac{1}{h_x^2} + \frac{1}{h_y^2} \end{bmatrix}_{(N-1)\times(N-1)}$$

$$I_y = \begin{bmatrix} \frac{1}{h_y}^2 & & & & \\ & \frac{1}{h_y}^2 & & & \\ & & \frac{1}{h_y}^2 & & \\ & & ..... & & \\ & & & \frac{1}{h_y}^2 & \\ & & & & \frac{1}{h_y}^2 \end{bmatrix}_{(N-1)\times(N-1)} \quad v_i = \begin{bmatrix} v_{i,1} \\ v_{i,2} \\ v_{i,3} \\ ... \\ v_{i,N-2} \\ v_{i,N-1} \end{bmatrix}_{(N-1)\times 1} \quad f_i = \begin{bmatrix} f_{i,1} \\ f_{i,2} \\ f_{i,3} \\ ... \\ f_{i,N-2} \\ f_{i,N-1} \end{bmatrix}_{(N-1)\times 1}$$

3 Classical Iterative Methods

There are multiple iterative methods that could approximate the solution of the linear system $Au = f$, such as Jacobi method, Gauss-Seidel method, red-black Gauss-Seidel and Successive Over-Relaxation method. Here we mainly discuss weighted Jacobi method.

Weighted Jacobi method starts with an initial guess of the solution, and then improves the accuracy of the solution through iteration. For the problem in Equation (4), the iterative approximation is produced through first solving the (i, j) unknowns by holding the current approximation of $(i, j-1)$, $(i, j+1)$, $(i-1, j)$, $(i+1, j)$ still, then add the current approximation of (i, j) by weight to the new approximation of point (i, j). It produces solution that converges to the exact solution by smoothing the errors. The iterative method could be written as

$$v_{ij}^{(new)} = (1-w)v_{ij}^{(old)} + \frac{w}{4}(v_{i-1,j}^{(old)} + v_{i+1,j}^{(old)} + v_{i,j-1}^{(old)} + v_{i,j+1}^{(old)} + h^2 f_{ij}), \quad 1 \le i, j \le N-1 \tag{5}$$

4 Geometric Multigrid

Though classical iterative methods are easy to implement and reduce high frequency errors efficiently, a major limitation of these methods is that they reduce low frequency errors slowly. The idea of multigrid method is to change the problem to a coarser grid, where low frequency errors become high frequency, and then we can use classical iterative method to efficiently remove the low frequency errors. In this way, we can substantially reduce the number of iterations that is required for the system to converge.

Before stepping into multigrid algorithm, we first introduce some key elements: Prolongation/Interpolation, Restriction and Relaxation. Consider the linear system $Au = f$ where u is the exact solution of the differential equation, and v is an approximation to u. Denote $e = u - v$ as the error of the approximation, and $r = f - Av$ as the residual of the linear system.

4.1 Interpolation/Prolongation

Interpolation or prolongation in multigrid algorithm is to transfer from the coarse grid to fine grid. Let $\Omega^{2h} \in \mathbb{R}^{(N/2-1)\times(N/2-1)}$ denote the coarse grid space where grid distance is $2h$, and $\Omega^h \in \mathbb{R}^{(N-1)\times(N-1)}$ denote the fine grid space where grid distance is h. Let v^{2h} be the approximate solution on Ω^{2h}, and v^h be the approximate solution on Ω^h. Then, the interpolation is given by

$$I_{2h}^h : \Omega^{2h} \to \Omega^h, \quad v^h = I_{2h}^h v^{2h} \tag{6}$$

The most simple and commonly used interpolation method is *linear interpolation*, where we distribute the value of c-points (coarse level points) on to the f-points (fine level points) by weight, as shown in Figure 1. The red point is c-point, and the black ones are f-points. The data points on the coarse grid Ω^{2h} could be written as in Equation (7).

$$\begin{aligned}
v_{2i,2j}^h &= v_{i,j}^{2h} \\
v_{2i+1,2j}^h &= \frac{1}{2}(v_{i,j}^{2h} + v_{i+1,j}^{2h}) \\
v_{2i,2j+1}^h &= \frac{1}{2}(v_{i,j}^{2h} + v_{i,j+1}^{2h}) \\
v_{2i+1,2j+1}^h &= \frac{1}{4}(v_{i,j}^{2h} + v_{i+1,j}^{2h} + v_{i,j+1}^{2h} + v_{i+1,j+1}^{2h}), \quad 0 \le i, j \le \frac{N}{2} - 1
\end{aligned} \tag{7}$$

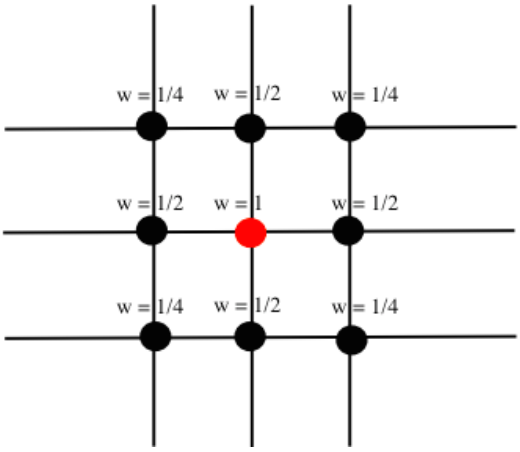


Figure 1: Two dimension linear interpolation/prolongation

4.2 The Restriction

The restriction in multigrid algorithm is to transfer residual from fine grid to coarse grid in order to further approximate the solution. Restriction is given by

$$I_h^{2h} : \Omega^h \to \Omega^{2h}, \quad v^{2h} = I_h^{2h} v^h \tag{8}$$

The most commonly used restriction methods are *restriction by injection* and *restriction by full weighting*.

Restriction by injection is the simplest restriction method. We directly take the value of the corresponding coarse grid value on the fine grid. It is given by

$$I_h^{2h} : \Omega^h \to \Omega^{2h}, \quad v_{i,j}^{2h} = v_{2i,2j}^h, \quad 1 \le i, j \le \frac{N}{2} - 1 \tag{9}$$

Restriction by full weighting is very similar as interpolation/prolongation in nature, instead of distribute value by weight, restriction by full weighting aggregate the adjunct data points by weights and forms a coarse grid. It takes the weighted average of f-points and puts value on the c-point, as shown in Figure 2. Same as previous, the red point is c-point, and the black ones are f-points. The data points on the fine grid Ω^h through restriction by full weighting could be written as in Equation (10).

$$\begin{aligned} v_{i,j}^{2h} =& \frac{1}{16}(v_{2i+1,2j+1}^h + v_{2i-1,2j+1}^h + v_{2i+1,2j-1}^h + v_{2i-1,2j-1}^h) \\ &+ \frac{1}{8}(v_{2i,2j+1}^h + v_{2i+1,2j}^h + v_{2i,2j-1}^h + v_{2i-1,2j}^h) \\ &+ \frac{1}{4} v_{2i,2j}^h, \quad 1 \le i, j \le \frac{N}{2} - 1 \end{aligned} \tag{10}$$

4.3 Relaxation

Relaxation is to apply classical iterative methods to remove high frequency errors. As introduced above, there are several methods such as Jacobi method, weighted Jacobi Method, Gauss-Seidel method, red-black Gauss-Seidel and Successive Over-Relaxation method.

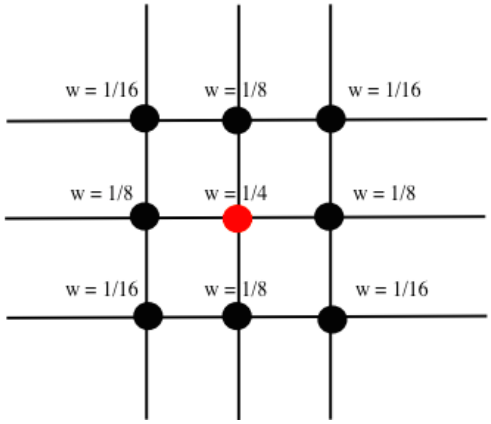
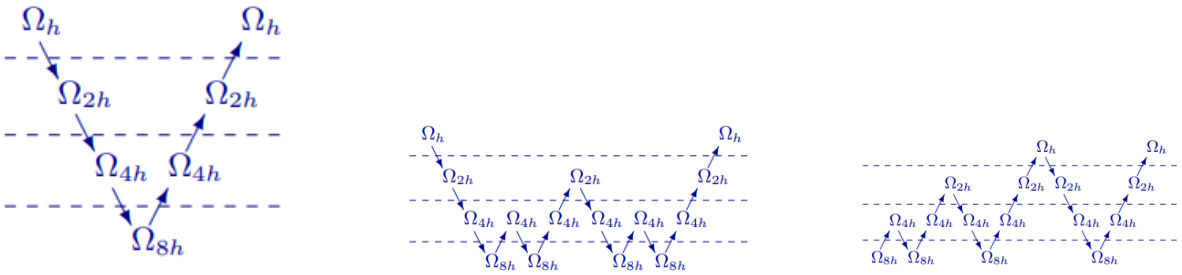


Figure 2: Two dimension restriction by full weighting

4.4 Multigrid Algorithm

The main idea of multigrid algorithm is to approximate the solution of linear system $Au = f$ through classical iterative method on the fine grid (pre-smoothing), then compute the residual $r = f - Av$, restrict the residual to the coarse grid, and continue to solve the linear system $Ae = r$ recursively. After we reach the preset level, we interpolate the coarse grid solution e to the fine grid, and add the interpolated e to the existing fine grid solution v, which corrects the fine grid solution. If necessary, we could perform post-smoothing on the linear system at this stage, which is applying classical iterative methods to further reduce high frequency errors, and improve the accuracy of the solution.

There are several commonly used multigrid schemes, such as V-cycle, W-cycle and Full Multigrid V-cycle (FMV-cycle), as shown in Figure 3. Here we give the detailed multigrid



(a) V-cycle (b) W-cycle (c) FMV cycle

Figure 3: Multigrid Algorithm Schemes

algorithm under the V-cycle scheme, the other schemes are similar. The V-cycle is shown in Figure 4. The recursive algorithm is shown in Algorithm 1.

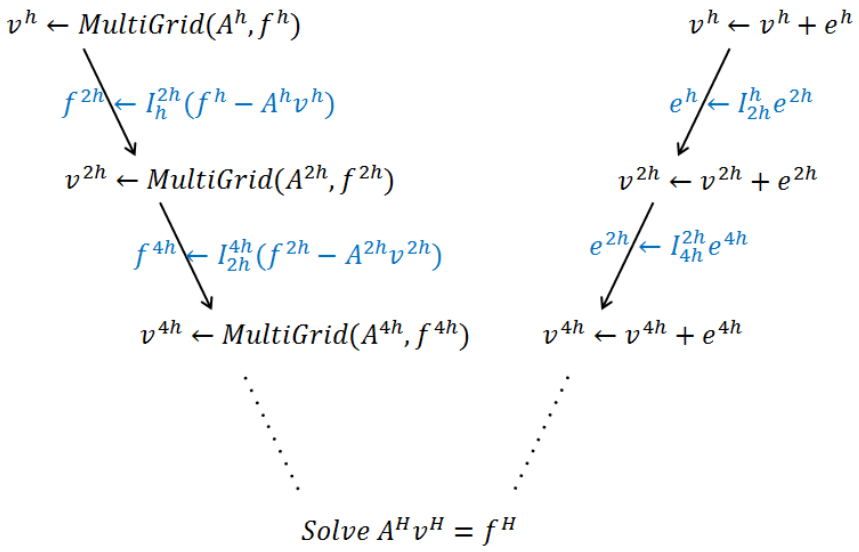


Figure 4: Multigrid V-cycle

Algorithm 1 Multigrid V-cycle

procedure MULTIGRID(f, v, l) ▷ l is the current multigrid level
  if $l = 1$ **then** ▷ Coarsest level
    return approximate/exact solution v
  end if
  $v \leftarrow$ PRESMOOTHING(v, f) ▷ Fine level pre-smoothing solution
  $r \leftarrow f - Av$ ▷ Fine level residual
  $r^c \leftarrow$ RESTRICTION(r) ▷ Coarse level residual
  $v^c \leftarrow 0$ ▷ Set coarse level initial solution guess to 0
  $e^c \leftarrow$ MULTIGRID($r^c, v^c, l - 1$) ▷ Recursive, coarse level solution (error term)
  $e \leftarrow$ INTERPOLATION(e^c) ▷ Fine level error term
  $v \leftarrow v + e$ ▷ Fine level corrected solution
  $v \leftarrow$ POSTSMOOTHING(v, f) ▷ Fine level post-smoothing solution
  return v
end procedure

5 A Test Problem

We test the performance of multigrid algorithm using a two dimensional Poisson problem, given by

$$-u_{xx} - u_{yy} = 2[(1-6x^2)y^2(1-y^2) + (1-6y^2)x^2(1-x^2)] \tag{11}$$

inside the unit square, with $u = 0$ on the boundary. The solution to this problem is

$$u(x, y) = (x^2 - x^4)(y^4 - y^2) \tag{12}$$

We test the multigrid algorithm on a 128×128 grid, and use multigrid V-cycle scheme to analyze the algorithm performance. The converge performance of multigrid algorithm is shown in Figure 5. We use weighted Jacobi method as the smoothing method for multigrid. The x-axis is the iteration times of the weighted Jacobi method, and the y-axis is the logarithm of the solution error under the Frobenius norm. The example solution plots after 1000 iterations are shown in Figure 6. The result shows multigrid converges way faster compared to the weighted Jacobi method, and the approximation is closer to the exact solution.

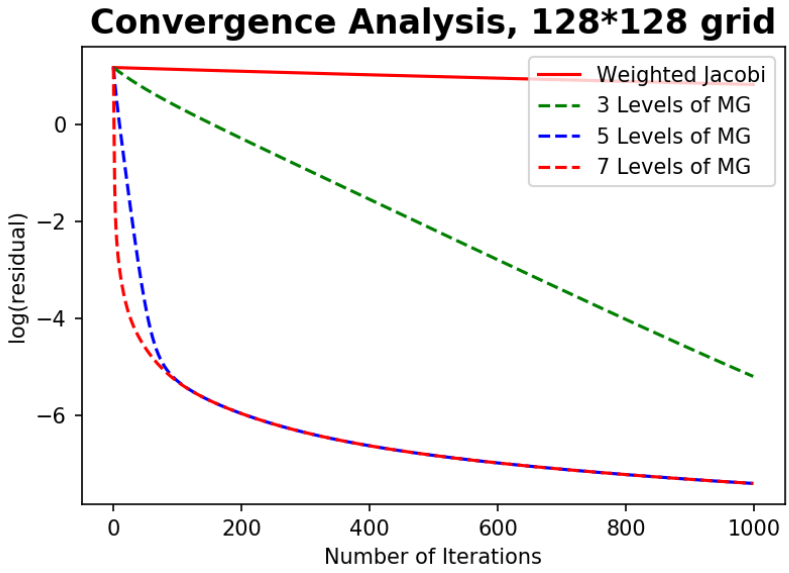


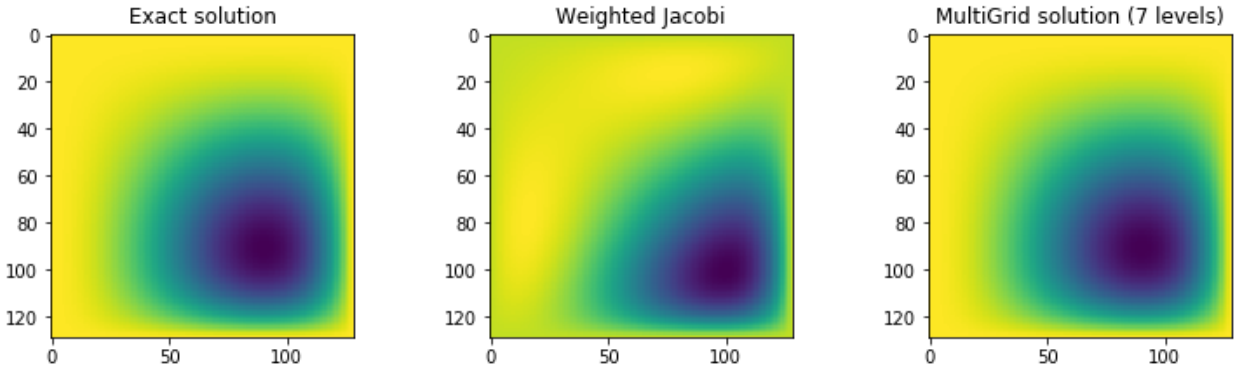
Figure 5: Examples of multigrid performance

6 Multigrid Algorithm in Fluid Simulation

In this section we discuss how multigrid algorithm is used in fluid simulation. We use Jos Stam's paper *Real-Time Fluid Dynamics for Games* as an example, which presents us a stable algorithm that simulates fluid flows.

Fluid simulations are based on the Navier-Stokes equations. The incompressible Navier-Stokes equations are given by

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \tag{13a}$$



(a) Exact Solution (b) Weighted Jacobi (c) 7 Levels of Multigrid

Figure 6: Example solution plots after 1000 iterations

$$\nabla \cdot \vec{u} = 0 \tag{13b}$$

where $\vec{u}$ is the flow velocity, ρ stands for the density of the fluid, p stands for pressure, $\vec{g}$ stands for body forces and ν stands for kinematic viscosity. Equation (13a) is the *momentum equation* and Equation (13b) is called *incompressibility condition*.

In the example paper, the Navier-Stokes equations that represent the velocity field (Eq. 14a) and density field (Eq. 14b) are given by

$$\frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{u} + \nu \nabla^2 \vec{u} + \vec{f} \tag{14a}$$

$$\frac{\partial \rho}{\partial t} = -(\vec{u} \cdot \nabla)\rho + \kappa \nabla^2 \rho + S \tag{14b}$$

The fluid is modeled on a $(N+2) \times (N+2)$ unit square grid ($N \times N$ interior), and assumes the density and velocity in each grid cell are constant.

There are three main operators used in the simulation: Diffusion, Advection and Projection.

Diffusion operator simulates the diffusion across the grid cells. The operator traces the densities backwards in time so that it will provide a stable output without oscillation when diffusion rate goes up. Set a as the diffusion rate, X_0 as the initial density and X the density after time step dt. Diffusion in cell (i, j) could be expressed as

$$X_0(i,j) = X(i,j) - a[X(i-1,j) + X(i+1,j) + X(i,j-1) + X(i,j+1)) - 4X(i,j)]$$

This forms a linear system $AX = X_0$, where X_0 is a known variable, and X is the unknown we are interested in. The operator A is very sparse and we could use classical iterative method (e.g. Gauss-Seidel method) to approximate the solution of the linear system.

Advection operator simulates the velocity field across the grid cells. For density, the operator treats densities as sets of moving particles, which lie in the center of grid cells. The problem is solved through tracing those particles backwards to their original place in the previous time step, which is before unit time step dt. Note that the currently centered particles in previous time step does not necessary be in the cell center, and linear interpolation method could be used to approximate the values.

Projection operator enforces the velocity to be mass conserving, which is a characteristic of incompressible. The operator decomposites the velocity field into mass conserving field and gradient field through Hodge decomposition, and the mass conserving field is obtained through subtracting the gradient field. The Hodge Decomposition algorithm first calculates

$$D(i,j) = -0.5 \times [u(i+1,j) - u(i-1,j) + v(i,j+1) - v(i,j-1)]$$

where u stores the horizontal component of the velocity field, and v stores the vertical component of the velocity field. Then we are interested the Poisson problem which has known variable matrix D and unknown P. The discretized format of the problem for cell (i,j) is

$$[P(i-1,j) + P(i+1,j) + P(i,j-1) + P(i,j+1)] - 4P(i,j) = D(i,j)/N^2$$

This form a linear system $AP = D$. We apply **multigrid algorithm** here, which will result in a more accurate and fast convergence result compared to the classical iterative method (Gauss-Seidel relaxation) used in the original paper. The final step is to get the mass conserving velocity field as

$$u(i,j) = u(i,j) - 0.5 \times [P(i+1,j) - P(i-1,j)]$$
$$v(i,j) = v(i,j) - 0.5 \times [P(i,j+1) - P(i,j-1)]$$

The density field could be simulated via the following steps:

$$\text{Add Source} \rightarrow \text{Diffuse} \rightarrow \text{Advect}$$

The velocity field could be simulated via the following steps:

$$\text{Add Source} \rightarrow \text{Diffuse} \rightarrow \text{Project} \rightarrow \text{Advect} \rightarrow \text{Project}$$

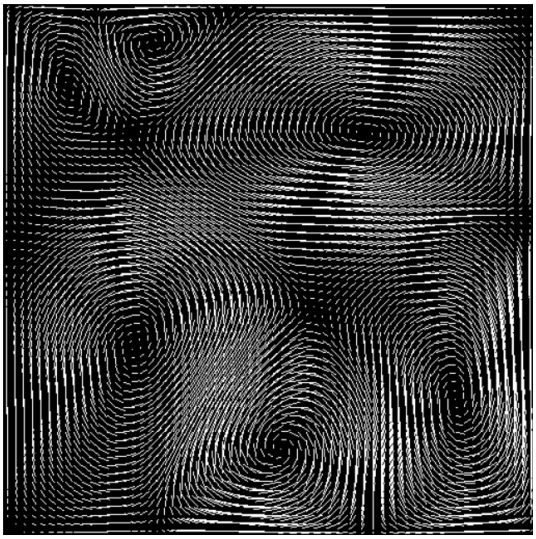


Figure 7: A snapshot of the velocity and density grid