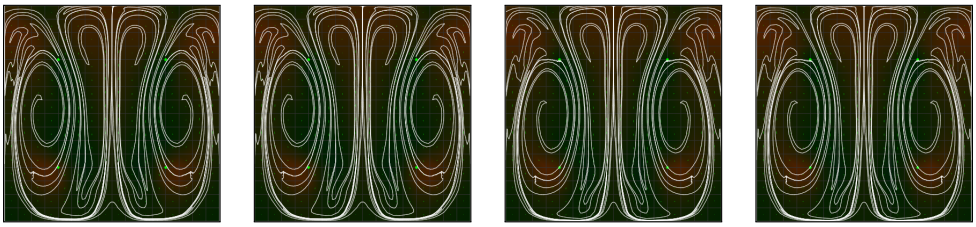
Poisson Solvers for Two-Dimensional Fluid Simulation in a Box

Joey Litalien* (260532617)
COMP 559 — Fundamentals of Computer Animation
McGill University, Winter 2017



(a) Gauss–Seidel relaxation (b) Successive over-relaxation (c) Conjugate gradient (d) Incomplete Cholesky conjugate gradient

Figure 1: *Solving the Discrete Poisson Problem on a MAC Grid.* Different numerical solvers are used on a 4-source scene simulation. The results are visually very similar, yet not identical for the same number of iterations per frame.

Abstract

In this project, we implement and benchmark four numerical solvers for the voxelized Poisson problem in two dimensions, namely Gauss–Seidel relaxation, successive over-relaxation, and conjugate gradient with and without an incomplete Cholesky preconditioner. Algorithms are tested on simple scenes with heat sources placed inside a MAC-discretized box to generate velocity fields.

Keywords: animation of fluids, Navier–Stokes, numerical solvers, implicit elliptic PDE solvers

1 Introduction and Related Work

Incompressible flows have proven very effective at generating many realistic and compelling effects in computer graphics including fluids such as water, rising smoke, and fire. One of the most popular methods for enforcing the incompressibility constraint is MAC grid based projection as first introduced by [Harlow and Welch 1965]. Since then, many efforts have been put into solving the Poisson equation for the cell centered pressures.

Enforcing incompressibility via the voxelized Poisson technique was originally popularized by [Foster and Metaxas 1996] who used successive over-relaxation (SOR) to solve the system. It was later shown by [Foster and Fedkiw 2001] that a preconditioned conjugate gradient algorithm with an incomplete Cholesky preconditioner was more efficient. This has since become a very widely adopted approach for solving the Poisson problem.

In this paper, we explore several numerical solvers to solve the discrete Poisson equation. After briefly reviewing the theory for simulating fluids, we analyze the behaviour of four algorithms, namely Gauss–Seidel relaxation, successive over-relaxation, and conjugate gradient with and without an incomplete Cholesky preconditioner. The solvers are benchmarked on three simple test scenes in a 2D box with solid boundary walls.

*Department of Electrical and Computer Engineering, joey.litalien@mail.mcgill.ca

2 Background

2.1 Governing Equations

Fluid flows are governed by the Navier–Stokes equations which describe the motion of viscous fluid substances. This is expressed by the momentum equation

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

where $\mathbf{u}$ is the fluid velocity, $\mathbf{f}$ is the body forces, ν is the kinematic viscosity, ρ is the fluid density and p is the pressure. More precisely, the term $(\mathbf{u}\cdot\nabla)$ represents self-advection and models how the velocity field flows within itself. The forces $\mathbf{f}$ are external forces such as gravity, wind or user interaction. The term $\nu\nabla^2\mathbf{u}$ represents diffusion and simulates the viscosity of the fluid.

Since changes in volume are rarely noticeable, we require the fluid to be incompressible and thus divergence-free:

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

Hence, the last term $-\nabla p/\rho$ in (1) is the subtraction of the pressure gradient. This comes from the Helmholtz–Hodge decomposition of the vector field when enforcing the incompressibility condition (2).

2.2 MAC Discretization

To solve the Navier–Stokes equations numerically, we use the stable fluids method from [Stam 1999] to separate the momentum equation into its components and solve them separately. In particular, the pressure component amounts to solving

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{1}{\rho}\nabla p = 0, \qquad \nabla\cdot\mathbf{u} = 0. \tag{3}$$

The classical approach to space discretization is to use a MAC (Marker and Cell). The 2D MAC scheme employs a uniform $(N+2)\times(N+2)$ grid with cell width $h \in (0,1)$. The inner domain is defined as $\Omega = \{(x,y) : h \le x, y \le hN\}$ with unit boundary $\partial\Omega$. Pressures p are stored at cell centers (i,j), while velocities are decomposed in two Cartesian components (u,v) and stored at cell

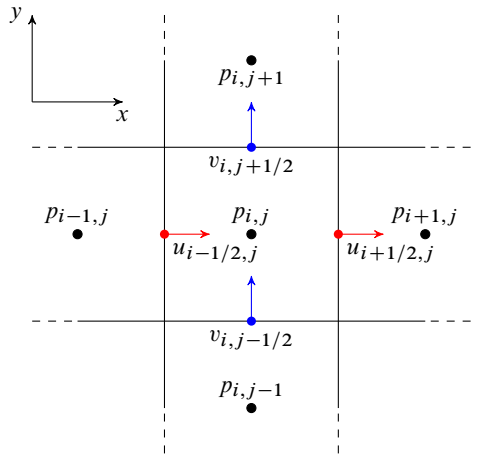


Figure 2: *Two-Dimensional MAC Grid.* The values (p, u, v) are stored at different location on the grid.

faces. This 5-point finite difference stencil is illustrated in Figure 2.

To enforce the Dirichlet and Neumann boundary conditions respectively, we set $\mathbf{u} \equiv \mathbf{0}$ on $\partial\Omega$ and let $\partial p/\partial \hat{\mathbf{n}} = \nabla p \cdot \hat{\mathbf{n}}$, where $\hat{\mathbf{n}}$ is the normal of the derivative of p.

The pressure at grid voxel (i, j) can be updated using the central difference approximations for $\partial p/\partial x$ and $\partial p/\partial y$ in a simple Forward Euler stepping:

$$u^{n+1}_{i+1/2,j} = u_{i+1/2,j} - \delta\left(\frac{1}{\rho}\frac{p_{i+1,j} - p_{i,j}}{h}\right) \tag{4}$$

$$v^{n+1}_{i,j+1/2} = v_{i,j+1/2} - \delta\left(\frac{1}{\rho}\frac{p_{i,j+1} - p_{i,j}}{h}\right) \tag{5}$$

where δ is the time step. The divergence can be similarly approximated as

$$(\nabla \cdot \mathbf{u})_{i,j} = \frac{u_{i+1/2,j} - u_{i-1/2,j}}{h} + \frac{v_{i,j+1/2} - v_{i,j-1/2}}{h}. \tag{6}$$

To enforce the incompressibility condition, we substitute (4) and (5) into (6), set it to zero and rearrange the terms to obtain:

$$\frac{\delta}{\rho}\left(\frac{4p_{i,j} - p_{i+1,j} - p_{i,j+1} - p_{i-1,j} - p_{i,j-1}}{h^2}\right) = -\left(\frac{u_{i+1/2,j} - u_{i-1/2,j}}{h} + \frac{v_{i,j+1/2} - v_{i,j-1/2}}{h}\right). \tag{7}$$

This gives a numerical approximation to the elliptic partial differential equation

$$-\frac{\delta}{\rho}(\nabla^2 p)_{i,j} = -(\nabla \cdot \mathbf{u})_{i,j}. \tag{8}$$

2.3 Voxelized Poisson Problem

Equation (8) is known as the **discrete Poisson problem**, a finite difference numerical method to discretize the 2D Poisson equation $\nabla^2\varphi = f$. Dividing by $-\delta/\rho$ yields a linear system with one equation per cell of the form

$$\mathbf{Ap} = \mathbf{d}, \tag{9}$$

where $\mathbf{p}$ is a vector containing all pressures $p_{i,j}$, $\mathbf{d}$ is a vector containing all negative divergences $(\nabla \cdot \mathbf{u})_{i,j}$ and $\mathbf{A}$ is a large, sparse coefficient matrix known as the 5-point Laplacian matrix. More precisely, $\mathbf{A}$ is an $N^2 \times N^2$ block tridiagonal matrix of the form

$$\mathbf{A} = \begin{bmatrix} D & -I & 0 & 0 & 0 & \cdots & 0 \\ -I & D & -I & 0 & 0 & \cdots & 0 \\ 0 & -I & D & -I & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & -I & D & -I & 0 \\ 0 & \cdots & 0 & 0 & -I & D & -I \\ 0 & \cdots & 0 & 0 & 0 & -I & D \end{bmatrix}, \tag{10}$$

where I is the $N \times N$ identity matrix and D is $N \times N$ tridiagonal with

$$D = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & \cdots & 0 \\ -1 & 4 & -1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 4 & -1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & -1 & 4 & -1 & 0 \\ 0 & \cdots & 0 & 0 & -1 & 4 & -1 \\ 0 & \cdots & 0 & 0 & 0 & -1 & 4 \end{bmatrix}, \tag{11}$$

Since $\mathbf{A}$ is symmetric positive definite,[1] it can be efficiently stored in memory as only the diagonal and the lower (or upper) part of $\mathbf{A}$ need to be saved to reconstruct the full matrix.

Here we assume that $\rho = 1$ for simplicity. Note that in this representation, the divergence vector must be stacked column-wise, that is, if $(\nabla \cdot \mathbf{u})_{i,j} = g_{ij}$ then

$$\mathbf{d} = -h^2(g_{11}, g_{21}, \ldots, g_{N1}, g_{12}, g_{22}, \ldots, g_{N2}, \ldots, g_{NN})^\top. \tag{12}$$

3 Numerical Solvers

There are several algorithms for solving symmetric positive definite linear systems. In this paper, we evaluate the performance of some of the most popular numerical solvers for this problem, namely

- Gauss–Seidel relaxation (GSR)
- Successive over-relaxation (SOR)
- Conjugate gradient (CG)
- Conjugate gradient with incomplete Cholesky preconditioner (CGIC)

3.1 Relaxation Iterative Methods

The simplest approach in solving the discrete Poisson problem is to use a Gauss–Seidel relaxation, where for each voxel (i, j) we solve for $p^{(k)}_{i,j}$ by using the current neighbouring pressure values and divergence $d_{i,j}$:

$$p^{(k)}_{i,j} = \frac{1}{4}\big(d_{i,j} + p_{i+1,j} + p_{i,j+1} + p_{i-1,j} + p_{i,j-1}\big). \tag{13}$$

The successive over-relaxation is similar to GSR, but uses the value of $p_{i,j}$ weighted by an over-relaxation parameter ω that can be tuned to optimize the convergence:

$$p^{(k)}_{i,j} = (1-\omega)p_{i,j} + \frac{\omega}{4}\big(d_{i,j} + p_{i+1,j} + p_{i,j+1} + p_{i-1,j} + p_{i,j-1}\big). \tag{14}$$

[1] If there is some fluid region entirely surrounded by solid walls with no empty air cells, $\mathbf{A}$ is semi-positive definite. We will only focus on the all fluid cell case.

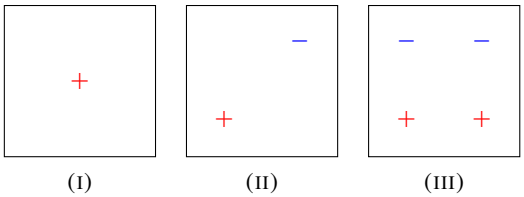


Figure 3: *Test Scenes on* Ω. Red: $\theta = +1.0$, Blue: $\theta = -1.0$.

It can be shown (see [Demmel 1997]) that SOR(ω) converges only for values of $\omega \in (0, 2)$ and achieves better performance than Gauss–Seidel on the interval $(1, 2)$.

3.2 Conjugate Gradient Methods

Conjugate gradient algorithms are well-suited for the Poisson problem since $\mathbf{A}$ is sparse, symmetric and positive-definite. The standard algorithm is used, as well as its preconditioned version. The Incomplete Cholesky factorization decomposition $\mathbf{M}^{-1} = \mathbf{K}\mathbf{K}^*$ is obtained as a preconditioner to solve $\mathbf{M}^{-1}(\mathbf{A}\mathbf{p} - \mathbf{d}) = \mathbf{0}$ instead, where $\mathbf{K} \approx \mathbf{L}$ is lower triangular and approximates the exact Cholesky decomposition $\mathbf{L}\mathbf{L}^*$. We find $\mathbf{M}$ by using algorithm to compute the exact Cholesky decomposition, except that any entry is set to zero if the corresponding entry in $\mathbf{A}$ is also zero.

4 Implementation and Setup

4.1 Algorithm Implementation

Algorithms are implemented in Java using the code from Assignment IV as a starting point. The initial non-staggered grid structure is first converted to a MAC scheme, and the sparse matrix $\mathbf{A}$ is formed based on the grid dimension. All grid cells are considered as fluid cells. Appropriate modifications at solid wall boundaries are done after solving the linear systems and are thus not accounted for when assembling the divergence vector $\mathbf{d}$. The JBLAS libary routines are used for linear algreba operations (*e.g.*, multiply, dot product) and results are stored as FloatMatrix. This imposes serious limitations on the implementation as JBLAS does not support sparse matrices.

4.2 Simulation Setup

We simulate very simple test scenes in a 2D box with static sources and assess how each algorithm performs. We run three different simulations for $\gamma = 60$ time units each. The three test scenes are presented in Figure 3 where $n = hN$ and θ is the source temperature.

When analyzing convergence rates of iterative solvers, a common measure is the norm squared of the residual error $\mathbf{r}^{(k)} = \mathbf{A}\mathbf{p} - \mathbf{d}$ at each iteration k of the algorithm. This method is intuitive when solving a single linear system, but ambiguous in this context since the Poisson equation is solved numerous times. Moreover, measures like clock wall time directly depend on the implementation and cannot be used in the current settings. Number of iterations needed for convergence is an interesting measure, but cannot be employed to contrast solutions generated by GSR or SOR for instance. This measure is computed and displayed, but not used in the study.

To compare algorithms for each scene on a time domain $\mathcal{T} = [0, \gamma]$, we run Gauss–Seidel with a large number of iterations ($K \gg 1000$) to obtain a "ground truth"[2] solution for the pressure vector $\widetilde{\mathbf{p}}_t$ at each time step t. We then use mean square error (MSE) to evaluate the difference in pressure with this baseline solution:

$$\text{Err}(\mathbf{p}) = \frac{1}{\Delta T} \sum_{t=0}^{\Delta T} \| \widetilde{\mathbf{p}}_t - \mathbf{p}_t \|^2, \tag{15}$$

where $T = \gamma / \Delta t$. Each simulation is ran with parameters $\gamma = 60$ and $\delta = 0.1$, and we evaluate results on a 16×16 inner grid Ω for simplicity. The error tolerance for the two conjugate gradient algorithms is set to $\epsilon = 1.0 \times 10^{-6}$ and we use $K = 20$ iterations for all solvers. An over-relaxation parameter of $\omega = 1.5$ is used for SOR(ω).

5 Results

Results for all benchmarks are plotted in PGFPlots and presented in Figure 4 below.

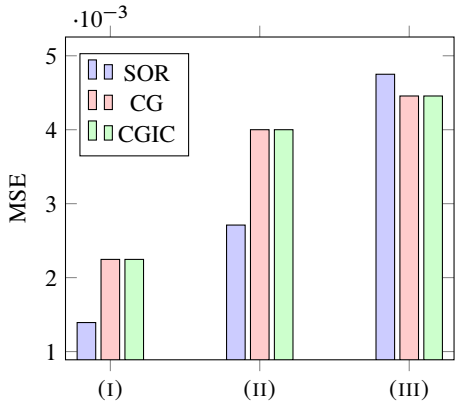


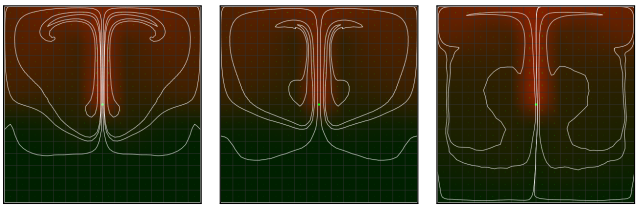
Figure 4: *Mean Square Error on Simulated Scenes.*

We immediately observe that for the simpler test cases (I) and (II), SOR(ω) performs better than the conjugate gradient solvers. However, CG has lower MSE on the more complex scene (III). While the MSE for incomplete Cholesky is consistently lower than its standard counterpart, both gives very similar results.

Visually, it is very hard to discern any difference between all numerical solvers with equal parameters. Asymmetry and graphical variations can be well noticed for a high tolerance ϵ and low number of iterations K for conjugate gradient, as shown in Figure 5. More examples can be found in the attached video where all cases are compared side by side, with $K = 30$.

From our experiments, successive over-relaxation is the best suited algorithm for simpler scenes like the one we benchmarked. However, these tests only scratch the surface when it comes to truly investigate convergence rates. The incomplete Cholesky preconditioned conjugate gradient hints towards a better scaling overall, which is consistent with results from the literature [McAdams et al. 2010]. This partially explains why it is still considered the state-of-the-art solution for the voxelized Poisson problem, and why it is often used as the go-to solver in most fluid simulators.

[2] Until Navier–Stokes equations are solved there is no such thing as "ground truth", but it is close enough to the exact solution.



(a) Gauss–Seidel, reference (b) Conjugate gradient, $\epsilon = 0.001$, $K = 4$ (c) Conjugate gradient, $\epsilon = 0.1$, $K = 1$

Figure 5: *Simulation Output for Scene* (I) *at* $t = 45$.

6 Conclusion and future directions

6.1 Discussion

In this paper, we implemented multiple numerical methods to solve for pressures in the Navier–Stokes equation. A MAC grid scheme was used to discretize space, and each component of the momentum equation was solved numerically using the stable fluids technique. Solutions to the discrete Poisson problem were found via relaxation iterative methods (Gauss–Seidel and successive over-relaxation) and sparse linear system solvers (conjugate gradient with and without incomplete Cholesky preconditioning). Algorithms were tested on three simple scenes, and mean square error was used as a measure of comparison.

6.2 Method Analysis

In hindsight, thoroughly studying the convergence of all algorithms implemented would have been more beneficial in assessing their true performance, space and time wise. Due to limitations in the sparse matrix implementation, it was difficult to truly benchmark all algorithms and understand their performance. Using the Matrix Toolkit Java (MTJ) instead of JBLAS could have been a better route to explore due to its structured sparse matrix support, thus allowing us to use computation time as an effective measure. It is also unclear whether using GSR with large number of iterations as ground truth is indeed an effective way of comparing the algorithms.

6.3 Future work

In the future, it would be interesting to implement more numerical solvers and benchmark them on the different scenes while varying the sources position in time, for example. User interaction could also be used to modified the velocity field in real-time to create more scenes. Using different values of N for the same scene would be a valuable exercise as well. Comparing incomplete Cholesky and modified incomplete Cholesky preconditioners could give insights on how the algorithms behave and scale in general.

The next logical step for this project would be to implement multigrid and its relatively recent conjugate gradient variations suggested by [McAdams et al. 2010]. Other commerical sparse linear systems solvers such as PARDISO [Schenk and Gärtner 2004] and SuperLU [Li 2005] could also be benchmarked in 2D, while making sure to account for any overhead introduced when calling C++ methods inside Java.

7 Acknowledgments

Special thanks to Keven Villeneuve and Sayantan Datta for suggesting mean square error as a measure of comparison between solvers.

References

BRIDSON, R. 2016. *Fluid Simulation for Computer Graphics*, 2nd ed. CRC Press.

DEMMEL, J. W. 1997. *Applied Numerical Linear Algebra*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, ch. Iterative Methods for Linear Systems.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, SIGGRAPH '01, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. In *Proceedings of the Conference on Graphics Interface '96*, Canadian Information Processing Society, GI '96, 204–212.

HARLOW, F. H., AND WELCH, J. E. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. In *Physics of Fluids*, vol. 8, 2182–2189.

LI, X. S. 2005. An overview of SuperLU: Algorithms, implementation, and user interface. In *TOMS*, vol. 31, 302–325.

MCADAMS, A., SIFAKIS, E., AND TERAN, J. 2010. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 65–74.

SCHENK, O., AND GÄRTNER, K. 2004. Solving unsymmetric sparse systems of linear equations with PARDISO. In *Journal of Future Generation Computer Systems*, vol. 30, 475–487.

STAM, J. 1999. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., SIGGRAPH '99, 121–128.