A Fast Unsmoothed Aggregation Algebraic Multigrid Framework for the Large-Scale Simulation of Incompressible Flow

HAN SHAO*, LIBO HUANG*, and DOMINIK L. MICHELS, KAUST

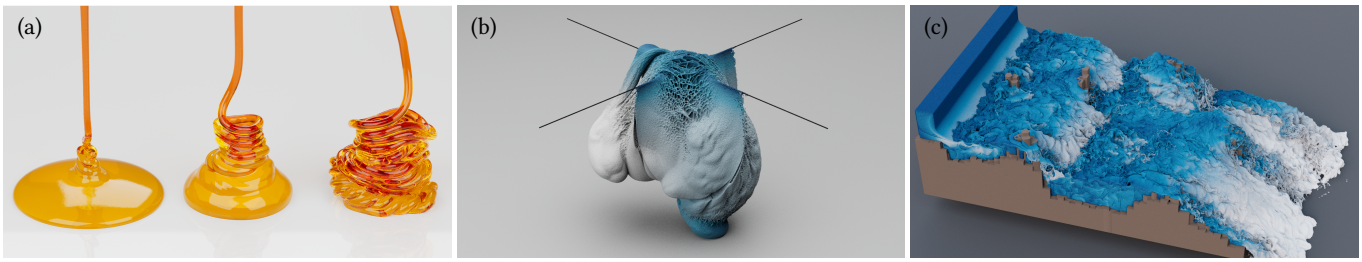


Fig. 1. Our framework enables the fast simulation of (a) buckling and coiling effects for liquids with various viscosity, (b) highly viscous liquids colliding with thin wires, and (c) the dynamics of non-viscous fluids interacting with a complex riverbed geometry.

Multigrid methods are quite efficient for solving the pressure Poisson equation in simulations of incompressible flow. However, for viscous liquids, geometric multigrid turned out to be less efficient for solving the variational viscosity equation. In this contribution, we present an Unsmoothed Aggregation Algebraic MultiGrid (UAAMG) method with a multi-color Gauss-Seidel smoother, which consistently solves the variational viscosity equation in a few iterations for various material parameters. Moreover, we augment the OpenVDB data structure with Intel SIMD intrinsic functions to perform sparse matrix-vector multiplications efficiently on all multigrid levels. Our framework is 2.0 to 14.6 times faster compared to the state-of-the-art adaptive octree solver in commercial software for the large-scale simulation of both non-viscous and viscous flow. The code is available at http://computationalsciences.org/publications/shao-2022-multigrid.html.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: Fluid Dynamics; Multigrid Methods; Single Instruction, Multiple Data (SIMD); Viscous Liquids.

ACM Reference Format:
Han Shao, Libo Huang, and Dominik L. Michels. 2022. A Fast Unsmoothed Aggregation Algebraic Multigrid Framework for the Large-Scale Simulation of Incompressible Flow. *ACM Trans. Graph.* 41, 4, Article 49 (July 2022), 18 pages. https://doi.org/10.1145/3528223.3530109

1 INTRODUCTION

Liquids are ubiquitous in our daily life ranging from rivers and water falls to honey and syrup. Efficient simulations of fluid phenomena have been extensively studied for physically-based animation. Eulerian and hybrid Lagrangian-Eulerian methods on regular Cartesian grids [Foster and Fedkiw 2001; Stam 1999; Zhu and Bridson 2005] are widely adopted for large-scale fluid simulations in visual computing. The most time consuming part of these methods was believed to be the projection step [Chorin 1967] where a pressure Poisson equation is solved. Viscosity also plays an important role in fluid simulation. Based on above methods, the decoupled variational formulation of viscosity [Batty and Bridson 2008] introduces viscosity with accurate free surface boundary conditions. Solving the extra variational viscosity equation is an order of magnitude slower than solving the pressure Poisson equation. In this paper, we focus on the most common one-way coupled free surface flow driven by prescribed velocity boundary conditions. Specifically, we solve the pressure Poisson equation and variational viscosity equation efficiently with another multigrid method.

Solving an equation efficiently can be achieved by solving with fewer unknowns, or solving faster with the same unknowns. Spatially adaptive methods effectively reduce the degrees of freedom (DOF) in the Poisson equation and viscosity equation [Aanjaneya et al. 2017; Ando and Batty 2020; Goldade et al. 2019; Losasso et al. 2005, 2004]. However special care must be taken to keep the spatial order of accuracy and the symmetry of the matrix so that efficient preconditioned conjugate gradient can be used. For these two equations, we observed a neglected potential of the regular Cartesian grid. The continuous memory layout of the DOFs not only reduces cache misses in the matrix-vector multiplication in an iterative solver, but also allows multiple rows of the result vector to be calculated with a single instruction. This is called single instruction, multiple data (SIMD). This potential was exploited in elasticity problems [Mitchell et al. 2016]. Later on, Liu et al. [2018] used SIMD to accelerate the sparse matrix-vector multiplication in a multigrid preconditioned conjugate gradient solver for topology optimization. We are inspired by their work to use SIMD to accelerate matrix-vector multiplications on regular Cartesian grids, and use multigrid to reduce the total number of multiplications. Ideally, SIMD can achieve 4× to

*Both authors contributed equally to this project.

Authors' address: Han Shao, han.shao@kaust.edu.sa; Libo Huang, libo.huang@kaust.edu.sa; Dominik L. Michels, dominik.michels@kaust.edu.sa, KAUST, Visual Computing Center, Thuwal 23955, KSA.

0730-0301/2022/7-ART49
https://doi.org/10.1145/3528223.3530109

16× speed up depending on available instruction sets, which is comparable to the reduction factor of DOFs in adaptive methods. This motivates us to conduct research on SIMD accelerated multigrid for Poisson equation and viscosity equation. However, the memory bandwidth for retrieving matrix coefficients and vector values limits the actual speed up of SIMD instructions. Inspired by McAdams et al. [2010] who used compact representations of the matrix coefficients, we use a similar mechanism that only keeps the non-trivial coefficients near liquid boundaries. We load the default trivial matrix coefficients where possible, for example in the interior of the liquid, so that all CPU cores fetch the same constant coefficients from cache lines.

Apparently, such a SIMD technique can be applied to all levels in geometric multigrid [Chentanez and Müller 2011; McAdams et al. 2010]. However, geometric multigrid (GMG) necessitates a careful design of coarse level discretizations for good convergence, and is limited to specific spatial discretizations. Therefore we turn our attention to the more versatile algebraic multigrid (AMG) which is less explored in fluid simulation. In AMG, a coarse level matrix is the product of coarse-fine interpolation matrices and the immediate fine level matrix (the *Galerkin principle*). In general, the coarse level matrix does not maintain the sparsity pattern. Fortunately, we find that the Unsmoothed Aggregation Algebraic MultiGrid (UAAMG) [Stüben 2001] maintains the stencil pattern for the 7-point stencil in the pressure Poisson equation, and the 15-point stencil in the variational viscosity equation on all multigrid levels. This allows us to apply the same efficient SIMD accelerated matrix-vector multiplication subroutine for all levels. Analyzing the CPU usage we found that the coarsening process is mostly serial if we use the explicit sparse matrix format in the Eigen library [Guennebaud et al. 2010], either due to the serial memory allocation of a large continuous space, or a serial implementation of the multiplication. We propose to store the matrix coefficients in OpenVDB floating point grids [Museth 2013] and calculate the coarse level matrix with a parallel implementation.

A key component in multigrid is an efficient smoother to reduce the high frequency errors on each multigrid level. The variational viscosity equation is not diagonally dominant for large viscosity, so the damped Jacobi smoother is not guaranteed to converge. Although the matrix is not diagonally dominant, it is positive definite after regularization, making the Gauss-Seidel method attractive. However, the Gauss-Seidel method is serial in nature for general linear systems. Luckily, we observe that the DOFs of the variational viscosity equations can be easily categorized into six colors according to the parity of the sum of the 3D indices. The diagonal term in each row has a different color than all other off-diagonal terms. The DOFs of the same color are updated concurrently, and different colors are updated sequentially.

Based on these observations, we develop an UAAMG framework for both the pressure Poisson equation and the variational viscosity equation. Important features of this framework include:

- a fast matrix-free sparse matrix-vector multiplication algorithm based on Advanced Vector Extensions (AVX) instructions and default coefficient trimming which is about 3× faster than parallel row-major sparse matrix-vector multiplication;
- a stencil-preserving Galerkin coarsening strategy that builds the coarse level matrix purely based on the spatial distribution of DOFs and matrix coefficients on the fine level, without special treatment of the boundary;
- a parallel matrix-free implementation of the coarsening strategy that is 4× to 10× faster than explicit sparse matrix-matrix multiplications using the Eigen library [Guennebaud et al. 2010];
- a multi-color Gauss-Seidel smoother for the variational viscosity equation that is easy to implement and parallel in nature.

In the following sections, we present our framework with rich illustrations of our multigrid algorithm, detailed analysis of the Poisson solver and viscosity solver for various configurations, and the performance of the whole simulation pipeline in complex scenes. Compared to the state-of-the-art adaptive variational viscosity solver [Goldade et al. 2019] implemented in Houdini [Side Effects Software 2021], our viscosity solver is about 3× faster. Our Poisson solver is about 2.5× faster than the adaptive octree pressure projection in Houdini, which is a variant of Lossaso's octree solver [Losasso et al. 2005, 2004]. Overall the total run time is about 3× faster per-substep compared to Houdini.

2 RELATED WORK

Popular fluid solvers in computer graphics include Eulerian [Enright et al. 2002; Foster and Fedkiw 2001; Stam 1999] and Hybrid Lagrangian-Eulerian methods [Jiang et al. 2015; Zhu and Bridson 2005], Lagrangian particles such as smoothed-particle hydrodynamics (SPH) [Bender and Koschier 2017; Ihmsen et al. 2014a,b; Müller-Fischer et al. 2003; Solenthaler and Pajarola 2009], and Lattice Boltzman methods [Lyu et al. 2021; Thürey and Rüde 2009]. For a more thorough introduction to fluid solvers we refer to the course notes of Bridson and Müller-Fischer [2007], Koschier et al. [2019], and Bridson's textbook [2016]. In this section, we mostly focus on multigrid methods in fluid simulation and within viscosity solvers.

2.1 Multigrid

We refer the reader to Briggs et al. [2000] for a very nice introduction to multigrid methods. We summarize some multigrid methods in Table 1. For the pressure Poisson equation, UAAMG is not new. Stüben [2001] wrote a concise review of the algebraic multigrid approach including such aggregation-based variant. Although it converges slower than the classical AMG [Ruge and Stüben 1987], there are simple fixes that make UAAMG as efficient as the classical AMG for scalar Poisson problems. Zarifi [2020] used aggregation based algebraic multigrid for smoke simulation but did not discuss the possibility of such a method for liquid simulation. We tried to use the V cycle variant in its supplemental material for liquid simulation, but it did not work well. It tried to implement smoothed aggregation in the V cycle by including extra relaxations after piecewise constant interpolation. However it did not include such relaxations when building coarse level matrices, which was unorthodox compared to the work of Vaněk et al. [2001]. Bolz et al. [2003] applied the multigrid method to 2D fluid simulation. They used piecewise linear interpolations and built coarse levels based on the Galerkin principle.

Table 1. Summary of different multigrid methods. 1st and 2nd refers to the spatial order of accuracy.

Method	Problem	Discretization	Matrix Coarsening	DOF Coarsening	Interpolation	Smoother
Vaněk et al. [1996]	General elliptic	General	Galerkin	Aggregation by strong off-diagonal	Smoothed constant	Gauss Seidel
Bolz et al. [2003]	2D Gas	Homogeneous boundary, 5-pt FD	Galerkin	Vertex 4-to-1	Linear	Damped Jacobi
Molemaker et al. [2008]	Gas	1st Dirichlet, 7-pt FD	Re-discretization	Vertex 8-to-1	Linear	Red-black SOR
McAdams et al. [2010]	Liquid	1st Dirichlet/Neumann 7-pt FD	Re-discretization	Voxel 8-to-1	Linear	Damped Jacobi
Chentanez and Müller [2011]	Liquid	2nd Dirichlet, 1st Neumann, 7-pt FD	Re-discretization	Voxel 8-to-1	Linear	Red-black GS
Ferstl et al. [2014]	Liquid	2nd Dirichlet/Neumann, adaptive FEM	Galerkin	Element 8-to-1 topology aware	Linear	Multi-color GS
Setaluri et al. [2014]	Gas	1st Neumann, Adaptive FD	Re-discretization	Voxel 8-to-1	Linear	Damped Jacobi
Dick et al. [2015]	Liquid	2nd Neumann, 7-pt FD	Galerkin	Vertex 8-to-1 topology aware	Linear	Multi-color GS
Weber et al. [2015]	Liquid	1st Dirichlet, 2nd Neumann, 7-pt FD	Re-discretization	Voxel 8-to-1	Constant	Red-black GS
Aanjaneya et al. [2017]	Liquid	2nd Dirichlet/Neumann, adaptive FD	Re-discretization	Voxel 8-to-1	Linear	Damped Jacobi
Aanjaneya et al. [2019]	Viscosity	Variational viscosity, 15-pt FD	Re-discretization	Voxel 8-to-1	Linear	Local Cholesky
Zarifi [2020]	Gas	1st Dirichlet, 7-pt FD	Galerkin	Voxel 8-to-1	Constant	Damped Jacobi
Our work	Liquid Viscosity	2nd Dirichlet/Neumann, 7-pt FD Variational viscosity, 15-pt FD	Galerkin	Voxel 8 to-1	Constant	Multi-color GS

Their linear interpolation expands the 7-point stencil pattern on the finest level to nine points to all coarse levels. Ferstl et al. [2014] applied a topology aware multigrid method with Galerkin principle to adaptive finite element simulation of free surface flow. However, their finite element formulation has inherently larger stencils, and they reported about 2× slower results than a 7-point finite difference formulation.

There are also plenty of geometric multigrid methods that build coarse level matrices directly by discretizing the problem on coarse levels. One of the most frequently mentioned geometric multigrid methods for pressure projection in free-surface flow simulations is the work of McAdams et al. [2010]. However it only supports the voxelized Poisson equation, which is limited to grid-aligned free surface and solid boundaries. Setaluri et al. [2014] extended this method to adaptive octree for smoke simulations. Aanjaneya et al. [2017] further extended the above adaptive octree for the simulation of liquids, and improved it to handle second order accurate boundary conditions. Chentanez and Müller [2011] adopted a similar geometric multigrid approach, but they use the volume fraction for solid boundary [Batty et al. 2007]. They used piecewise linear interpolation instead of piecewise constant interpolation for prolongation. They further adapted multigrid to handle separating boundary conditions [Chentanez and Müller-Fischer 2012]. Molemaker et al. [2008] applied multigrid methods to simple domains without obstacles, and instead handled complex boundaries with iterative orthogonal projections. Dick et al. [2015] proposed a geometric multigrid method for curved boundaries. Their targeted uncommon cases where the simulation domain is full of thin solid boundaries. In such scenarios, a coarse level DOF is split if it is restricted from disjoint fine level DOFs across thin boundaries or air gaps. Without such thin obstacles, their topology-aware improvements are not significant.

UAAMG shares similarities with GMG from [Weber et al. 2015] in solving the pressure Poisson equation. Both methods use Ng et al. [2009] strategy for second order solid boundary (Neumann boundary condition), 1-to-8 piecewise constant interpolation, and matrix-free multiplication. However, Weber et al. [2015] only showed first order convergence on their Dirichlet boundary, while our method achieves second order [Gibou et al. 2002], and theoretically works beyond liquid simulation. The most interesting coincidence is that both methods have the same off-diagonal terms for solid boundaries on all levels. The off-diagonal term in their method is proportional to the face liquid fraction which is averaged from 4 children face fractions. In our method, the off-diagonal term is the scaled average of four children off-diagonal terms. According to Ng et al. [2009], the off-diagonal term is proportional to face fraction.

Beside pressure projection, multigrids are also adopted in other applications in graphics. For example, Xian et al. [2019] used multigrid with piecewise constant interpolation for real-time simulation of elastic objects. They emphasized the importance of preserving sparsity of coarse level matrices. Tamstorf et al. [2015] applied smoothed aggregation multigrid to cloth simulation. Wang et al. [2018] developed a full multigrid method for cloth simulation. Zhu et al. [2010] used geometric multigrid with distributive Gauss-Seidel relaxation for simulation of elastic solids. Wang et al. [2020] proposed a time integration scheme for the material point method which has a multigrid solver at its core. Liu et al. [2018] combined SIMD operations and Galerkin coarsening based multigrid for narrow band topology optimization. Our work draws much inspiration from Liu et al. [2018]in terms of SIMD acceleration and the multi-color Gauss-Seidel smoother.

2.2 Viscosity

Recently, Larionov et al. [2017] proposed a unified variational pressure-velocity formulation for the simulation of viscous Newtonian liquids. Their method can better preserve surface details and produce better coiling behavior. However, their method is three times slower than the decoupled variational formulation of Batty and Bridson [2008]. Therefore, they concluded that the decoupled variational viscosity is still recommended if excessive surface smoothing is tolerable. For more discussion of the viscosity formulation please also see the references therein. Aanjaneya et al. [2019] tried to use a geometric multigrid method to solve the decoupled variational viscosity equation. However, they quickly discovered that traditional damped Jacobi smoother does not work well for this problem. They proposed to use a box-smoother with local Cholesky factorization. However, it has large overhead and the final pipeline was not quite advantageous compared to an incomplete Cholesky factorization

preconditioned conjugate gradient (ICPCG) method. Goldade et al. [2019] discretized the viscosity equation adaptively on an octree. Their method was up to 9× faster than the uniform discretization for the linear solve due to (up to 6×) reduced number of unknowns and (up to 2×) fewer iterations in the diagonal preconditioned conjugate gradient method. This adaptive viscosity solver was incorporated into Houdini [Side Effects Software 2021] and serves as the baseline of our comparison.

3 FLUID EQUATIONS AND DISCRETIZATION

The control equations for incompressible viscous fluid consist of the continuity equation,

$$\nabla \cdot \vec{\mathbf{u}} = 0\,, \tag{1}$$

and the momentum equation,

$$\vec{\mathbf{u}}_t + \left(\vec{\mathbf{u}} \cdot \nabla\right)\vec{\mathbf{u}} = -\frac{\nabla p}{\rho} + \frac{\nabla \cdot \tau}{\rho} + \vec{\mathbf{g}}\,, \tag{2}$$

where $\vec{\mathbf{u}}_t$ is the partial time derivative of the velocity $\vec{\mathbf{u}}$, p is the pressure, ρ is the density, $\vec{\mathbf{g}}$ is the gravitational acceleration and τ is the viscous shear stress tensor. The constitutive equation between the shear stress tensor and the velocity is given as

$$\tau = \mu\left(\nabla\vec{\mathbf{u}} + \nabla\vec{\mathbf{u}}^T\right),$$

where μ is the dynamic viscosity coefficient. At the free surface, we have $\tau \cdot \vec{\mathbf{n}} = 0$. At the solid boundary, a non-slip boundary condition $\vec{\mathbf{u}} = \vec{\mathbf{u}}_{\mathrm{BC}}$ is imposed.

We follow Batty and Bridson [2008] to solve Eq. (1) and (2), where the viscosity solver is sandwiched by two pressure projections. First, the advection is done and external forces are applied:

$$\frac{\vec{\mathbf{u}}^* - \vec{\mathbf{u}}^n}{\Delta t} + \left(\vec{\mathbf{u}}^n \cdot \nabla\right)\vec{\mathbf{u}}^n = \vec{\mathbf{g}}^n\,. \tag{3}$$

We follow the workflow of the incompressible Fluid-Implicit-Particle (FLIP) method [Zhu and Bridson 2005] to handle the advection step in Eq. (3). We use the midpoint rule for the time integration.
Then the pressure Poisson equation is solved,

$$-\nabla^2 p = \frac{\rho}{\Delta t}\nabla \cdot \vec{\mathbf{u}}^*\,, \tag{4}$$

and the velocity field is updated to be divergence-free:

$$\frac{\vec{\mathbf{u}}^{**} - \vec{\mathbf{u}}^*}{\Delta t} = -\frac{\nabla p}{\rho}\,. \tag{5}$$

The viscous effect is addressed by solving the following equation:

$$\frac{\vec{\mathbf{u}}^{***} - \vec{\mathbf{u}}^{**}}{\Delta t} = \frac{\nabla \cdot \tau}{\rho}\,. \tag{6}$$

Finally, the divergence is removed again by solving Eq. (4) and (5).

We will briefly review the discretization of the pressure Poisson Eq. (4) and the variational viscosity Eq. (6). We refer the reader to the open source code of Batty [2018] for a reference implementation of the viscous liquid simulator. Our contribution is not a novel discretization of the equation, but a novel solver on an already discretized system.

3.1 Pressure Poisson Equation Discretization

The pressure Poisson Eq. (4) is discretized with a 7-point stencil on a regular Cartesian grid. We use the cut-cell method of Ng et al. [2009] to handle irregular liquid-solid boundaries. For the liquid-air boundary, we use the ghost fluid method [Enright et al. 2002; Gibou et al. 2002] to achieve second order accuracy. In short, the off-diagonal term between two neighbor pressure DOFs as well as the right hand side divergence contribution based on the face velocity are weighted by the liquid face fraction. Near the free surface, an additional value is added to the diagonal term inversely proportional to the distance of the DOF to the free surface, leading to a pressure value close to zero near the free surface. Please see Figure 3 for an illustration of the matrix coefficients.

3.2 Variational Viscosity Equation Discretization

Batty and Bridson [2008] proposed a variational interpretation of Eq. (6). In this formulation, the traction free boundary condition is automatically enforced. Here, we use $\vec{\mathbf{u}}$ and $\vec{\mathbf{u}}^{\mathrm{old}}$ to replace $\vec{\mathbf{u}}^{***}$ and $\vec{\mathbf{u}}^{**}$:

$$\min_{\vec{\mathbf{u}}} \iiint \rho \left\|\vec{\mathbf{u}} - \vec{\mathbf{u}}^{\mathrm{old}}\right\|_2^2 + 2\Delta t \iiint \mu \left\|\frac{\nabla\vec{\mathbf{u}} + (\nabla\vec{\mathbf{u}})^T}{2}\right\|_F^2. \tag{7}$$

The minimization of the smooth energy in Eq. (7) leads to a symmetric and positive semi-definite system. In the discretized form, the minimal value in Eq. (7) is obtained when the following equations are satisfied:

$$\begin{aligned}
uV_u - \frac{\Delta t}{\rho}\left((V_p\tau_{xx})_x + (V_{\tau_{xy}}\tau_{xy})_y + (V_{\tau_{xz}}\tau_{xz})_z\right) &= u^{\mathrm{old}}V_u\,,\\
vV_v - \frac{\Delta t}{\rho}\left((V_{\tau_{xy}}\tau_{xy})_x + (V_p\tau_{yy})_y + (V_{\tau_{yz}}\tau_{yz})_z\right) &= v^{\mathrm{old}}V_v\,,\\
wV_w - \frac{\Delta t}{\rho}\left((V_{\tau_{xz}}\tau_{xz})_x + (V_{\tau_{yz}}\tau_{yz})_y + (V_p\tau_{zz})_z\right) &= w^{\mathrm{old}}V_w\,.
\end{aligned} \tag{8}$$

$$\tau_{xx} = 2\mu\frac{\partial u}{\partial x}\,,\ \tau_{yy} = 2\mu\frac{\partial v}{\partial y}\,,\ \tau_{zz} = 2\mu\frac{\partial w}{\partial z}\,,$$

$$\tau_{xy} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right),\ \tau_{yz} = \mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right),\ \tau_{xz} = \mu\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right).$$

In above equations, u, v, w are the three components of the velocity which are face-centered, and the stress components are either voxel-centered $(\tau_{xx}, \tau_{yy}, \tau_{zz})$ or edge-centered $(\tau_{xy}, \tau_{yz}, \tau_{xz})$. Their discrete values are calculated by finite differences.

The V terms refer to the volume fractions centered at different positions. V_p is cell-centered, V_u, V_v, V_w are face-centered and $V_{\tau_{xy}}, V_{\tau_{yz}}, V_{\tau_{xz}}$ are edge-centered. The volume fractions are determined by averaging their eight subcells, following the approach of Batty [2018], and Takahashi and Lin [2019]. A face velocity component is a DOF if it has non-zero velocity volume fraction, or its neighbor edge-centered volume or cell-centered volume is non-zero, meaning it is involved in at least one row of the equation. Once a face velocity is marked as a DOF, we set a minimum threshold for its control volume fraction, for example 0.1. This effectively adds threshold values to small diagonal terms, serving as a regularization to the potentially singular equation system. After modifying the minimum velocity control volume we then calculate the matrix

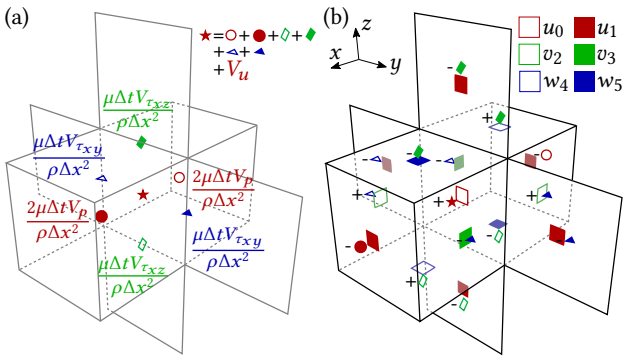


Fig. 2. Illustration of the matrix coefficients, stencil pattern, and update color for one row of the variational viscosity Eq. (8). (a) Matrix coefficients evaluated based on viscosity and stress control volumes located at edge centers and voxel centers, and velocity control volume located at the face center. (b) The stencil pattern of one row of Eq. (8) corresponding to the u component, as well as the values of the stencil pattern. The stencil involves 15 DOF, categorized into 6 colors for the multi-color Gauss-Seidel smoother, based on the parity of the sum of their spatial indices.

coefficients and right hand side values in Eq. (8). In the Appendix we show how this control volume regularization alleviates the singularity problem in a minimum 2D example.

We set up an equation for every DOF. Each row of the matrix has up to 15 non-zero terms. We show the matrix coefficients in Figure 2 (a). In Figure 2 (b), we show the spatial locations of the 15 DOFs in one row of the equation as colored square patches on voxel faces. They are color filled or empty depending on the parity of the sum of indices. The 15 DOFs are categorized into six colors used for the multi-color Gauss-Seidel smoother in multigrid. It consists of seven similar component terms (u, red square patches), and eight cross component terms (v and w, green and blue square patches). We also put corresponding matrix coefficient as star, circles, rhombuses, and triangles near each DOF square patch. A cross section view of the stencil for component u with staggered grid index (i, j, k) is presented in the Appendix for a more detailed reference.

4 UNSMOOTHED AGGREGATION MULTIGRID

Pressure Poisson and variational viscosity equations are solved by multigrid methods. Multigrid methods solve a system of equations through a hierarchy of matrices where level zero is the finest level. Typically it starts with the discretization of an equation on a grid with spatial resolution h:

$$A_h u_*^h = f^h\,,$$

where A_h is the matrix obtained through finite difference, finite volume or other methods, u_*^h is the exact solution, and f^h is the right hand side term. In multigrid, starting with an initial guess u^h, a few relaxation methods such as Jacobi or Gauss-Seidel iterations are applied but they can only reduce high frequency errors. Assume after relaxations, we get a solution u_{old}^h. The error e_{old}^h is defined as the difference to the exact solution:

$$e_{\text{old}}^h = u_*^h - u_{\text{old}}^h\,.$$

The residual equation holds for the current error:

$$A_h e_{\text{old}}^h = r_{\text{old}}^h = f^h - A_h u_{\text{old}}^h\,,$$

where r_{old}^h is the fine level residual. Solving the above residual equation on the original fine grid is not easier, because low frequency errors are still hard to remove. In multigrid methods, the residual equation is instead solved on a coarse grid, where low frequency errors look like high frequencies again. The fine level residual is transferred (called restriction) to coarse grid, and the solved coarse error are transferred back (called prolongation) to correct the current solution. A restriction operator R denotes the first step:

$$r^H = R r_{\text{old}}^h\,,$$

where r^H denotes the residual on the coarse level. In classical AMG [Stüben 2001], the restriction operator is obtained by analyzing the matrix coefficients, which is important in anisotropic problems. In some aggregation based algebraic multigrid methods [Vaněk et al. 1996], matrix analysis is also required, where fine DOFs connected by large off-diagonal terms are grouped together to form a coarse DOF. Here, we deal with structured grid and isotropic problems, hence we can exploit the geometric information. For the Poisson equation, a voxel on the coarse level aggregates eight voxels on the fine level. A coarse level voxel is a DOF, if any of its eight children voxels is a DOF. Figure 3 shows how coarse level DOFs are defined in two dimensions. In three dimensions the restriction matrix is defined as

$$R_{ij} = \begin{cases} 1/8, & \text{if coarse voxel } i \text{ covers fine voxel } j, \\ 0, & \text{otherwise.} \end{cases}$$

Now we need to solve the residual equation on the coarse level. A key question is the definition of A_H, the coarse level matrix. Generating the coarse level matrix is called coarsening. In UAAMG, A_H in three dimensions is defined by

$$A_H := R A_h P \;, with\; P = 8R^T\,, \tag{9}$$

where P is the prolongation or interpolation operator. It is the scaled transpose of the restriction operator. Obtaining the coarse level matrix A_H through a RA_hP sandwich is called the Galerkin principle. On the coarse level, the residual equation is solved:

$$A_H e^H = r^H\,,$$

and the error is transferred back to correct the fine level solution:

$$u_{\text{new}}^h = u_{\text{old}}^h + \alpha P e^H\,.$$

A constant α is introduced in the prolongation step. The motivation is to improve the convergence of UAAMG in a multigrid preconditioned conjugate gradient algorithm. Stüben [2001] mentioned that setting $\alpha = 2$ can improve the convergence for Poisson equation where UAAMG is used as a preconditioner because the coarse matrix A_H obtained by Galerkin principle and piecewise constant interpolation tends to underestimate the error correction approximately by half. However, when UAAMG is used as a standalone iterative solver, $\alpha = 1$ should be used to avoid overshooting and instabilities.

4.1 Algorithm Overview

Stüben [2001] pointed out that UAAMG is better used in combination with preconditioned conjugate gradient, μ-cycle ([Briggs et al. 2000], Chapter 3), and matrix rescaling. We refer the reader to Algorithm 3 by McAdams et al. [2010] for multigrid preconditioned conjugate gradient. Here, we summarize the less common multigrid μ-cycle borrowed from Briggs et al. [2000].

By default we use a μ-cycle with $\mu = 2$, which is the W-cycle. $\mu = 1$ is the common V-cycle. We use $\nu = 2$ in the pre-smooth and post-smooth steps. We observe no benefits for $\mu > 2$ and $\nu > 3$. In the post-smooth step, the relaxation should be applied symmetrically, for example, the red-black Gauss-Seidel should be black-red Gauss-Seidel. In the presence of moving solid boundary, for example when a character walks in a pool, there can be enclose regions with pure Neumann boundary conditions. In such cases, the multigrid preconditioned conjugate gradient fails to converge. We detect such failure and revert to standalone multigrid cycles with at least 8 iterations.

Algorithm 1: Recursive μ-cycle.

Input: initial guess u_l, current level l, right hand side f_l
Output: updated u_l
if $l = nLevels - 1$ **then**
  solve $A_l u_l = f_l$ directly;
  return u_l;
end
apply ν times relaxations to $A_l u_l = f_l$; `/* pre-smooth */`
$r_l = f_l - A_l u_l$; `/* calculate this level residual */`
$r_{l+1} = R_l r_l$; `/* restriction */`
$e_{l+1} \leftarrow 0$;
apply μ times $e_{l+1} \leftarrow \mu\text{-Cycle}(e_{l+1}, l+1, r_{l+1})$;
if *Poisson and Preconditioner* **then**
  $u_l \leftarrow u_l + 2P_l e_{l+1}$; `/* prolongation */`
else
  $u_l \leftarrow u_l + P_l e_{l+1}$; `/* prolongation */`
end
apply ν times relaxations to $A_l u_l = f_l$; `/* post-smooth */`
return u_l;

4.2 Matrix-free Coarsening

To generate the coarse level matrix A_H in Eq. (9), two sparse matrix-matrix multiplications are required. While it is readily available in an existing sparse linear algebra library such as Eigen [Guennebaud et al. 2010], it is slow because the stock implementation is serial. Furthermore, to harness the power of modern CPU SIMD instruction sets, we store matrix coefficients in 3D OpenVDB grids instead of using an explicit sparse matrix format such as compressed sparse row. It becomes necessary to develop a matrix-free coarsening implementation tailored for our data structure.

Each pressure DOF is associated with an index triplet (i, j, k) and occupies the center of a voxel. The Poisson matrix is stored in one diagonal term grid, and three off-diagonal term grids. The diagonal term is trivially stored with the same index. The matrix coefficients between current DOF and three neighbor DOFs on the negative axis direction are also stored with index (i, j, k). Thanks to the symmetry of the Poisson matrix, there is no need to store the off-diagonal terms on the positive direction. Please see Figure 3 for a illustration in 2D.

Algorithm 2: Poisson Coarsening.

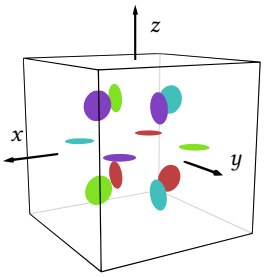
Input: fine level diagonal d_f, off-diagonals x_f, y_f, z_f
Output: coarse level diagonal d_c, off-diagonals x_c, y_c, z_c
for $ijk_c \in ActiveCoarseDOFGridVoxels$ **do**
  $d \leftarrow 0$; $x \leftarrow 0$; $y \leftarrow 0$; $z \leftarrow 0$;
  for $ijk_f \in ActiveDOFChildren(ijk_c)$ **do**
    $d \leftarrow d + d_\text{f}(ijk_\text{f})$; `/* affects self */`
    if $IsFineDOF(ijk_f - (1,0,0))$ **then**
      if $ijk_f.x = 2ijk_c.x$ **then**
        $x \leftarrow x + x_\text{f}(ijk_\text{f})$; `/* affects neighbor */`
      else
        $d \leftarrow d + 2x_\text{f}(ijk_\text{f})$; `/* affects self */`
      end
    end
    if $IsFineDOF(ijk_f - (0,1,0))$ **then**
      if $ijk_f.y = 2ijk_c.y$ **then**
        $y \leftarrow y + y_\text{f}(ijk_\text{f})$; `/* affects neighbor */`
      else
        $d \leftarrow d + 2y_\text{f}(ijk_\text{f})$; `/* affects self */`
      end
    end
    if $IsFineDOF(ijk_f - (0,0,1))$ **then**
      if $ijk_f.z = 2ijk_c.z$ **then**
        $z \leftarrow z + z_\text{f}(ijk_\text{f})$; `/* affects neighbor */`
      else
        $d \leftarrow d + 2z_\text{f}(ijk_\text{f})$; `/* affects self */`
      end
    end
  end
  $C = 1.0/8.0$;
  $d_\text{c}(ijk_\text{c}) \leftarrow C \cdot d$;
  $x_\text{c}(ijk_\text{c}) \leftarrow C \cdot x$;
  $y_\text{c}(ijk_\text{c}) \leftarrow C \cdot y$;
  $z_\text{c}(ijk_\text{c}) \leftarrow C \cdot z$;
end

The coarse level matrix is defined as $A_H = RA_hP$. The term between the ith DOF and jth DOF on coarse level is essentially how ith DOF's children DOFs on the fine level influence jth DOF's children DOFs through the fine level matrix A_h. We multiply both sides by e_i , the standard basis, in which the ith coarse level DOF is one and others are zeros, to get the ith column of A_H. This process is illustrated in the bottom row of Figure 3. With the input e_i, we set the ith DOF to be 1. Piecewise constant interpolation sets its children DOFs on the fine level to be 1, which gives Pe_i. Multiplying the fine level matrix A_h (conceptually, we never really invoke the

expensive function), we get $A_h P e_i$. Every non-zero fine level DOF in Pe_i contributes a corresponding column. Finally, fine level DOFs contributes to its coarse DOFs by simple averaging, resulting in $RA_h Pe_i$.

In practice we directly compute the diagonal term grid and three negative direction off-diagonal term grids. In 3D, the coarse diagonal term is 1/8 of the sum of all its children's diagonal terms plus two times of all off-diagonal terms between its children, because each child off-diagonal term contributes to two of its children. The coarse off-diagonal term at the interface of ith and jth voxel is 1/8 of the sum of off-diagonal terms between all ith voxel's children near the interface and jth voxel's children near the interface. Algorithm 2 shows this process.

Velocity DOFs are located at voxel faces. Each row of the matrix has seven terms between same-component DOFs, just like the Poisson matrix. There are three components, so this part of the viscosity matrix is stored in three diagonal terms and nine off-diagonal terms. Furthermore, each row of the matrix contains eight cross-component terms. There are three components, but again the matrix is symmetric, so there are $3 \times 8/2 = 12$ individual coefficients per voxel. We store these cross-component coefficient between six faces of a voxel with the same index (i, j, k) in 12 grids. In the inset figure, we plot the 12 coefficients as colored disks at the centers between six velocity DOFs. We use different colors to distinguish the terms. The same colors are used in Figure 4, where we show a 2D illustration of the coarsening process.



On the coarse level, the cross component terms in the viscosity equation are also calculated in a column by column fashion. We loop over three cross sections. Each cross section contains four cross component terms. The process is identical to the 2D case. In Figure 4 we show how the four terms in a coarse level voxel are calculated from cross component terms in its 4/8 children voxels in 2D/3D. The coarse level cross component term is simply a scaled summation of a part of its children's cross component terms. Please see Figure 4 for a visual illustration.

4.3 Smoothers

Multigrid relies on efficient smoothers to reduce high frequency errors on every level. We tried many smoothers for the Poisson equation, and found that scheduled relaxed Jacobi (SRJ) [Yang and Mittal 2017] is slightly better than others. For the viscosity equation, we found that only the multi-color Gauss-Seidel smoother works efficiently. The results are summarized in Table 2 and 3.

We began with the SPAI-0 smoother [Bröker et al. 2001], the default smoother of AMGCL [Demidov 2019], a general algebraic multigrid solver. SPAI-0 uses a diagonal matrix to approximate the inverse of the matrix in the linear equation. It is parameter free, efficient to apply, and has low overhead to compute the matrix. It is worth noting that Bröker et al. [2001] said SPAI-0 outperforms damped Jacobi with optimal weight in Poisson problems, which is $\frac{2d}{2d+1}$ where d is the dimension. However, as suggested in our experiments in Table 2, the SPAI-0 smoother is not as efficient as the damped Jacobi with $\omega = 6/7$. McAdams et al. [2010] proposed $\omega = 2/3$, but we found it less efficient compared to the optimal parameter 6/7 in 3D. SRJ is an extension to damped Jacobi with a single weight parameter. SRJ uses a different weight for every iteration, and finds the best weight combination. We found SRJ to be slightly faster than the damped Jacobi. We also tried the red-black SOR with slight over relaxation $\omega = 1.2$. It is slightly faster than red-black Gauss-Seidel.

Finding a proper smoother for the viscosity equation is indeed difficult as pointed out by Aanjaneya et al. [2019]. The slow convergence in damped Jacobi cannot be fixed by finding a proper weight. Sometimes it explodes. The parameter-free SPAI-0 also explodes sometimes. These difficult cases have large viscosity or small grid size Δx. As demonstrated in Figure 2, the matrix is diagonally dominant only if $\frac{\mu \Delta t}{\rho \Delta x^2}$ is small enough so that the sum of off-diagonal terms are smaller than the diagonal.

Fortunately, the multicolor Gauss-Seidel smoother works well for this problem. The velocity DOFs are colored according to which component they are and the parity of the sum of their indices (Figure 2 (b)).

5 SIMD-VDB ACCELERATION

Multigrid allows the equation to be solved with a small number of matrix-vector multiplications. Furthermore we use SIMD intrinsic functions to accelerate the computation, and use default matrix coefficients trimming to save limited memory bandwidth and improve computation throughput.

5.1 Data Structure

We store the DOFs of the equation, and the matrix coefficients on OpenVDB [Museth 2013]float grids. In OpenVDB, voxels are stored in leaf nodes. It provides flexible storage for data located sparsely in space. By default, each leaf node is dynamically allocated and contains $8 \times 8 \times 8 = 512$ voxels. In each leaf node, there is also a 512-bit bit mask that marks the active state of each voxel. In our Poisson solver, for example, if an index (i, j, k) is a pressure DOF, its containing leaf node is allocated, and the corresponding bit in the bit mask is 1. We record the DOFs' index in an OpenVDB int32 grid by using the parallel exclusive prefix sum algorithm. At the coarsest level, this index grid is used to convert the coefficient OpenVDB grid into the Eigen matrix, and is solved by its library.

5.2 Matrix-free Multiplication

When the matrix-vector multiplication is implemented in a matrix-free fashion, it involves accessing neighboring DOFs and corresponding matrix coefficients. The results are calculated concurrently in a row by row fashion. Usually the vector optimization is left for compilers.

The dimension of the leaf node is eight, which happens to be the length of an AVX SIMD vector. Therefore, each leaf node contains 64 AVX SIMD vectors that are well aligned in space. For the Laplace matrix, each DOF interacts with its six neighbors and itself. We load

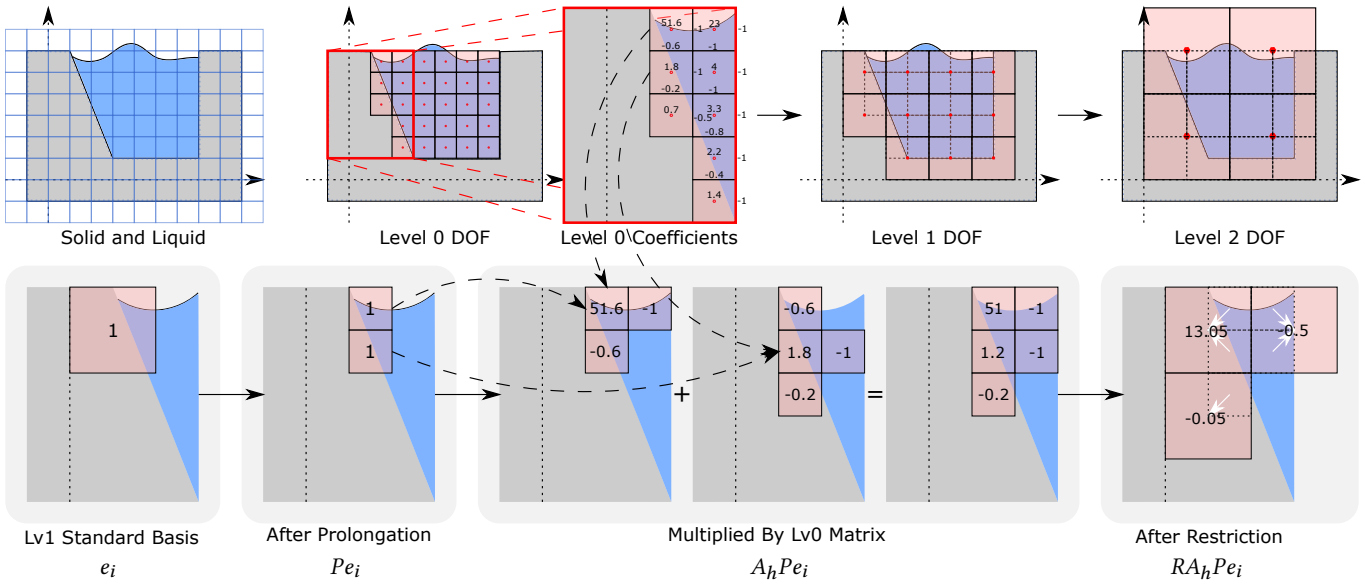


Fig. 3. 2D illustration of Galerkin coarsening operations for Poisson's equation. Top: simulation geometry and pressure DOF at each level. Active DOFs are denoted by red dots in red voxels. For each voxel we store the matrix diagonal term indicated by the number above the red dot, and matrix off-diagonal terms between neighbor DOFs indicated by numbers on the bottom (left) side. Bottom: calculating the level 1 matrix coefficients from level 0 coefficients. Each column of the level 1 matrix $A_{2h} = RA_hP$ is calculated in parallel. Starting with a standard basis e_i at level 1, we apply the prolongation operation Pe_i by setting its children at level 0 to value of 1. We use the level 0 matrix coefficients to calculate the intermediate level 0 stencil values A_hPe_i, and finally collect the level 0 stencil values to corresponding level 1 DOFs with results of RA_hPe_i. This gives us the level 1 coefficients and recursively those of deeper layers.

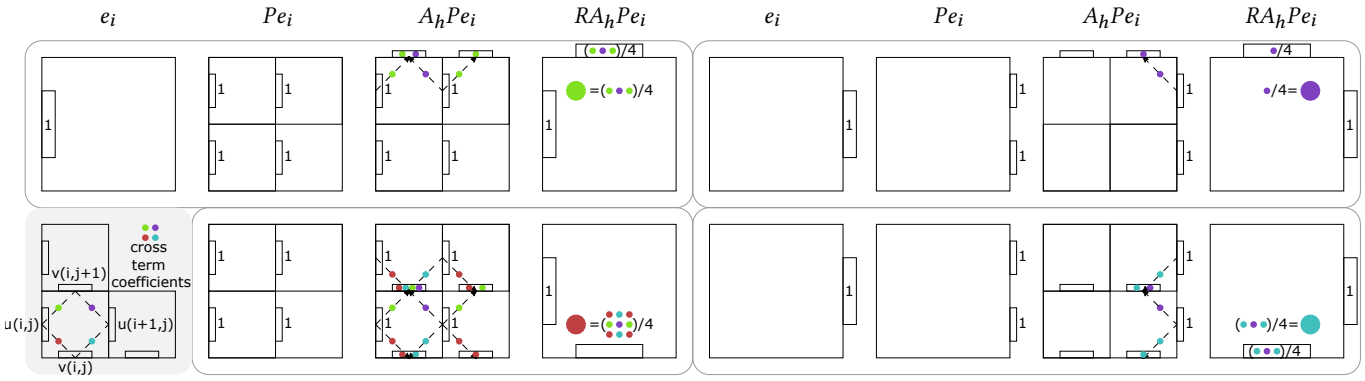


Fig. 4. 2D illustration of Galerkin coarsening operations for the cross component terms in the variational viscosity equation. Bottom left legend: each voxel stores the velocities of the bottom and left faces, and four cross component terms between velocities on four faces. On the coarse level, four cross terms can be calculated from those at immediate fine level. The input is a standard basis e_i at the coarse level. After prolongation we get Pe_i at the fine level. Using the cross terms at the fine level, we can calculate the stencil values A_hPe_i, and finally restrict the values to coarse level stencils RA_hPe_i. The resulting cross terms at the coarse level are scaled summations of the fine level cross terms. In three dimensions, 1/8 is used instead in restriction.

the seven neighboring DOF SIMD vectors, multiply corresponding matrix coefficients SIMD vectors, and add them together to get the resulting SIMD vectors. This process is illustrated in Figure 5. Before we write the result, we set the non-DOF voxels in the SIMD vector to zeros according to the active state bit mask.

In OpenVDB, SIMD vectors are organized along the z-direction. Neighbor SIMD vectors in x and y direction can be easily fetched by their starting addresses. Near the leaf boundary, we need to fetch SIMD vectors in neighbor leaf nodes multiple times. We cache pointers to first voxels in neighbour leaf nodes before all calculations to avoid frequent hash table inquiries.

Accessing neighbor SIMD vectors in the z-direction involves fetching SIMD vectors from two leaf nodes and merging them into one SIMD vector. We show related intrinsic functions and a visual illustration in Figure 6.

The variational viscosity equation has three components and extra cross component terms. The implementation is more laborious, but the building blocks are still caching neighbor leaf node pointers, fetching aligned SIMD vectors in the x and y neighbors, and data permutation for the z neighbor vectors. More discussions of the cross terms can be found in the Appendix.

In the relaxation step, the inverse of the diagonal is precomputed in a float grid to avoid division instruction. In the red-black or multi-color Gauss-Seidel, only the even or odd voxels are updated. This can be done by blending instructions with 01010101_2 as masks.

5.3 Default Coefficients Trimming

For both Poisson equation and viscosity equation, the diagonal term is the same in the interior of the liquid, and the off-diagonal terms are the same away from solids. This also applies for cross terms in the viscosity equation. It can happen that some coefficient leaves are filled with the same default value. When this happens, we delete the coefficient leaf node. Loading coefficient SIMD vectors from null pointers triggers loading default coefficients for corresponding terms. On the other hand, loading DOF SIMD vectors from null pointers returns all zeros.

Once the diagonal grid is trimmed, it no longer reflects the spatial distribution of the DOF. We use an integer grid to index each DOF. This index grid is never trimmed to keep the original spatial distribution of DOF. It is also the mapping from a 3D DOF grid to an 1D numeric vector.

6 UNIT TEST EXPERIMENTS

To evaluate the performance of our multigrid solver, we first conduct unit test experiments for solving the pressure Poisson equation and the variational viscosity equation. After the unit tests, we choose the best set of parameters to simulate various viscous and non-viscous fluids presented in the next section.

Figure 7 summarizes the unit test scenes. A sparse scene (Figure 7 (a)) and a compact scene (Figure 7 (b)) are designed for the pressure Poisson equation. For the variational viscosity equation, various viscosity coefficients can be used for the designed viscosity scene (Figure 7 (c)). In the unit test experiments, the convergence criterion is set according to a relative error tolerance of 10^{-7}. We compare various linear system solvers, Diagonal Preconditioned Conjugate Gradient (DPCG) with Eigen's implementation [Guennebaud et al. 2010], Incomplete Cholesky Preconditioned Conjugate Gradient (ICPCG) with Eigen's implementation, Batty's implementation [2018], and Unsmoothed Aggregation Algebraic MultiGrid (UAAMG) with explicitly built matrix approach and matrix-free SIMD-VDB approach, and the AMGCL library [Demidov 2019, 2020]. All the numerical experiments have been performed on a single workstation with 2 × 12-core Intel® Xeon® E5-2687W v4 CPUs and 256 GB DDR4-2133 memory.

6.1 Pressure Poisson Equation

As shown in Figure 7 (b), a compact scene is used to test the Poisson solver where an open tank is filled with water. We use the normalized index of DOF (0 to 1) plus a uniform noise from -1 to 0 as the right hand side. The former introduces long-wavelength and the latter introduces short-wavelength errors.

Table 2. Performance of the unit tests for the pressure Poisson equation in the compact scene. The resolution refers to the number of divisions along each axis. We compare DPCG, ICPCG with Eigen's implementation and Batty's implementation [Batty 2018], UAAMG with explicitly built matrix and matrix-free SIMD-VDB approach, and the AMGCL library [Demidov 2019, 2020]. Different smoothers are also compared, where DJ is damped Jacobi, SRJ is Scheduled Relaxed Jacobi, RBGS is Red-Black Gauss-Seidel and SPAI-0 is the Sparse Approximate Inverse-0 [Bröker et al. 2001]. The row with *SRJ, NT* for UAAMG (SIMD) is the result with No Trimming of default coefficients. In each of the box, we report the number of iterations towards convergence and the corresponding runtime. For algebraic multigrid based methods, we also report the time required to construct the matrix hierarchy inside the round parenthesis. The convergence criterion is set to be a relative error of 10^{-7} for all methods. The memory usage for UAAMG (SIMD) for different resolutions is 0.7 GB, 2.1 GB, 11.9 GB, and 84.7 GB respectively.

Resolution		128	256	512	1,024
# DOFs		2,146,685	16,974,589	135,005,693	1,076,890,621
DPCG		1,323 iters 18.6s	4,544 iters 584.7s	9,322 iters 8,756.4s	-
ICPCG (Eigen)		782 iters 108.9s	2,538 iters 6,407.1s	-	-
ICPCG (Batty)		195 iters 7.5s	395 iters 116.8s	903 iters 1,943.7s	-
UAAMG	DJ	11 iters 691ms (746ms)	11 iters 6.3s (6.4s)	11 iters 50.3s (53.2s)	-
	SRJ	9 iters 569ms (747ms)	9 iters 5.2s (6.5s)	9 iters 41.5s (53.7s)	-
	RBGS	7 iters 595ms (766ms)	7 iters 5.8s (6.5s)	8 iters 53.1s (54.6s)	-
	SPAI-0	12 iters 754ms (732ms)	12 iters 7.0s (7.0s)	13 iters 60.3s (54.7s)	-
UAAMG (SIMD)	DJ	11 iters 371ms (61ms)	11 iters 2,161ms (174ms)	11 iters 14.9s (935ms)	11 iters 111.6s (6.6s)
	SRJ	9 iters **308ms (60ms)**	9 iters **1,789ms (180ms)**	9 iters **12.3s (956ms)**	10 iters **102.9s (6.8s)**
	SRJ, NT	9 iters 376ms (60ms)	9 iters 2,534ms (198ms)	9 iters 18.7s (1,178ms)	10 iters 162.5s (8.9s)
	RBGS	7 iters 332ms (62ms)	7 iters 1,896ms (173ms)	8 iters 14.6s (915ms)	8 iters 108.7s (6.7s)
	SPAI-0	12 iters 424ms (58ms)	12 iters 2,457ms (181ms)	13 iters 18.2s (918ms)	13 iters 137.4s (6.7s)
AMGCL	DJ	18 iters 634ms (702ms)	24 iters 7.1s (4.8s)	26 iters 71.3s (40.6s)	-
	SPAI-0	17 iters 600ms (703ms)	23 iters 6.9s (5.0s)	25 iters 64.0s (38.8s)	-

Table 2 shows the performance report of various linear system solvers for the Poisson equation in the compact scene (Figure 7 (b)). Three baseline methods DPCG, ICPCG (Eigen), and ICPCG (Batty) are tested. ICPCG (Batty) achieves the best performance and is used as the baseline method for later sparse scene test (Figure 7 (a)). In 512^3 resolution, our UAAMG approach (SRJ smoother) with explicit matrix implementation already achieves 20.4× speedup over ICPCG (Batty) including the matrix hierarchy construction time. Our SIMD-VDB approach further achieves a speedup of 7.2× over the explicit matrix implementation, leading to an overall speedup of 147× over ICPCG (Batty). Using 128^3 and 256^3 resolutions, our framework achieves an overall speedup of 20× and 59× over ICPCG (Batty). It indicates that our framework has a much better performance especially in the high resolution cases which are usually needed for accurate and detailed fluid simulations. Using the resolution of 1024^3,

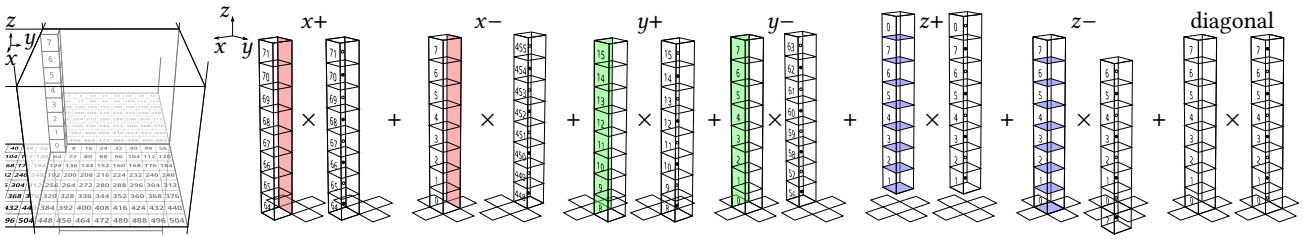


Fig. 5. The memory layout of voxels inside a $8 \times 8 \times 8$ leaf node, and illustration of a matrix-vector multiplication based on SIMD operations. Each leaf node is dynamically allocated and densely populated with voxels in a lexicographical order. The symmetric Laplacian matrix coefficients are stored in three off-diagonal terms indicated by voxels with red (x), green (y), blue (z) faces, and a diagonal term. The matrix-vector multiplication can be applied by multiplying corresponding matrix coefficient SIMD vectors with the DOF SIMD vectors and adding them together. The solid and hollow circles in each vector voxel indicate colors in the red-black Gauss-Seidel update. The number on each voxel is the memory offset to the first element in the corresponding leaf node. All SIMD vectors are native-continuous in the memory except the $z\pm$ SIMD vectors.

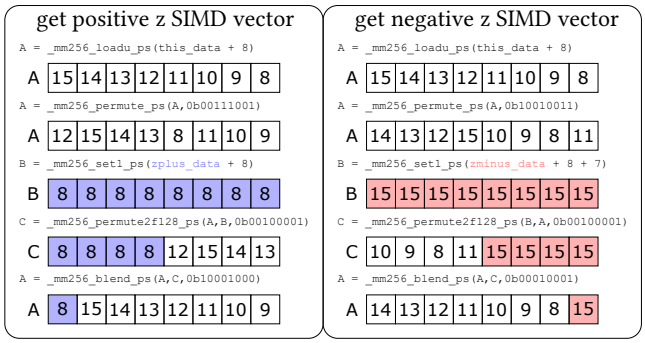


Fig. 6. Intrinsic functions for obtaining the positive/negative z vectors beginning with the **8th** voxel across two leaf nodes. The pointers to the first voxel of this leaf node, positive and negative neighbor leaf nodes are stored in `this_data`, `zplus_data`, and `zminus_data` respectively.

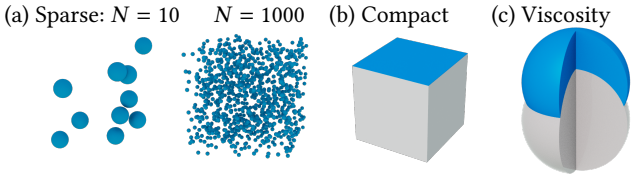


Fig. 7. Sketch of the unit test scenes for solving the pressure Poisson equation and the variational viscosity equation. The sizes of all scenes are 1 m^3. In (a), the overall number of DOFs is almost kept as a constant, while the physical shape is divided into different numbers of balls. For each individual ball, the boundary is a free surface, which results in a Dirichlet boundary condition for the pressure Poisson equation. In (b), a tank of liquid is used to construct the pressure equation. The top side is a free surface (Dirichlet boundary condition) and other faces are solid walls (Neumann boundary condition). In (c), a ball of viscous liquid is located on another solid ball. Please note that the cross section cut is just due to the visualization of the internal boundary. The free surface part of the liquid ball results in a Neumann boundary condition for the variational viscosity equation and the part contacting the solid ball results in a Dirichlet boundary condition.

only UAAMG (SIMD) has sufficient memory to solve the equation. Please note, that during the implementation, we trim default matrix coefficient nodes and replace them with default constant values as

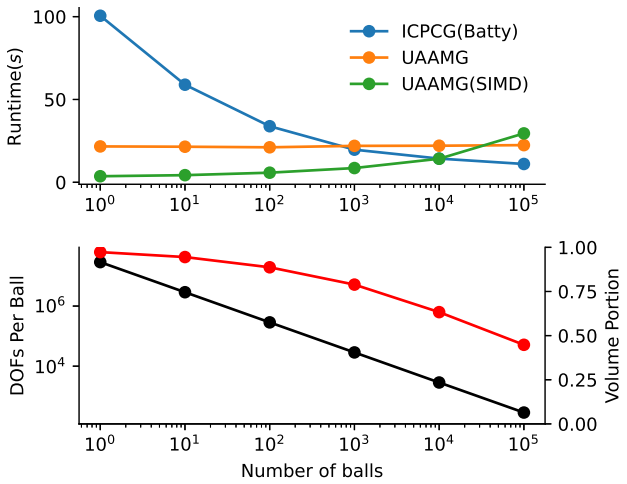


Fig. 8. Impact of sparsity within the fluid geometry on the solver's efficiency in the sparse scene in Figure 7 (a). The top figure presents the runtime with regard to the number of balls for three methods, ICPCG, UAAMG, and UAAMG (SIMD). In the bottom figure, the red line refers to the volume portion and the black line refers to the number of DOFs per ball. For all of the above tests, the total number of DOFs stays around 29 million. The SIMD approach has noticeable overhead for extremely disperse water drops, while ICPCG effectively solves thousands of small subsystems.

mentioned in Section 5.3. The comparison of NT (No Trim) with the final one shows that this technique alone achieves 1.5× speedup.

The compact scene is ideal for SIMD acceleration because almost all SIMD vectors are filled with DOFs. However, in the extremely sparse case where each SIMD vector only contain one DOF, the SIMD approach only introduces overhead. The sparse scene (Figure 7 (a)) is designed to systematically analyze the performance of SIMD acceleration as a function of sparsity. The number of balls is the parameter used to control the physical sparsity of the scene. A larger number of balls leads to a smaller droplet sizes with fixed total volumes. Figure 8 (top) shows that the SIMD approach is the fastest until 10^5 balls, which has 290 DOFs per ball. Multigrid with explicit matrix construction is insensitive to the sparsity pattern, while the

SIMD-VDB approach for the matrix multiplication introduces more overheads and eventually loses its advantage. Figure 8 (bottom) shows the volume portion based on the OpenVDB data structure. It is calculated as the total number of DOFs divided by 8 times the total number of non-empty SIMD vectors. The SIMD approach is favored unless in extremely sparse scenes.

6.2 Variational Viscosity Equation

Table 3. Performance of the unit tests for the variational viscosity equation. The resolution refers to the number of divisions along the radial direction. We have conducted experiments for two viscosity coefficients, $\mu = 1$ Pa · s and $\mu = 10^4$ Pa · s. Results for five methods, DPCG, ICPCG, UAAMG, UAAMG (SIMD) UAAMG (SIMD, NT) are presented. The multi-color Gauss-Seidel smoother is used for UAAMG, UAAMG (SIMD) and UAAMG (SIMD, NT). UAAMG (SIMD, NT) is designed for testing the performance with No Trimming of default coefficients. In each box, we report the number towards convergence and the corresponding runtime. For algebraic multigrid based methods, matrix hierarchy construction time are reported in the round parenthesis. The convergence criterion is set to be a relative error of 10^{-7} for all the methods. The memory usage for UAAMG (SIMD) of different resolutions is 1.2 GB, 3.8 GB, 10.1 GB, and 21.8 GB respectively.

Resolution		128	256	384	512
# DOFs		1, 892, 865	14, 627, 457	48, 805, 069	115, 018, 077
μ(1 Pa · s)	DPCG	33 iters **540ms**	65 iters 9.1s	98 iters 43.9s	133 iters 133.5s
	ICPCG (Batty)	19 iters 1,981ms	108 iters 52.5s	138 iters 216.4s	164 iters 572.5s
	UAAMG	6 iters 640ms (1,513ms)	9 iters 8.2s (11.4s)	10 iters 32.1s (40.4s)	11 iters 82.6s (100.8s)
	UAAMG (SIMD)	6 iters 417ms (454ms)	9 iters **3.3s (1.3s)**	10 iters **10.8s (3.4s)**	11 iters **26.1s (7.4s)**
	UAAMG (SIMD, NT)	6 iters 510ms (448ms)	9 iters 4.8s (1.2s)	10 iters 16.6s (3.0s)	11 iters 42.1s (6.6s)
$\mu(10^4$ Pa · s)	DPCG	1,155 iters 18.3s	2,705 iters 355.8s	4,706 iters 1,798.2s	5,375 iters 5,252.2s
	ICPCG (Batty)	484 iters 26.7s	1,186 iters 495.2s	2,154 iters 2,951.3s	-
	UAAMG	23 iters 2.3s (1.8s)	27 iters 24.1s (11.7s)	30 iters 94.8s (50.6s)	32 iters 242.1s (98.9s)
	UAAMG (SIMD)	23 iters **1.6s (451ms)**	27 iters **9.7s (1.3s)**	30 iters **31.8s (3.3s)**	32 iters **74.9s (7.5s)**
	UAAMG (SIMD, NT)	23 iters 1.9s (439ms)	27 iters 13.7s (1.2s)	30 iters 47.7s (3.0s)	32 iters 118.5s (6.7)

In Figure 7 (c) we place a spherical drop of viscous liquid on top of another solid sphere to test the solver for the variational viscosity Eq. (8). The r.h.s. of Eq. (8) requires the initialization of the velocity field. For each component of the velocity, we introduce three parts: the random noise with uniform distribution from -0.5 to 0.5, the relative position (normalized to -0.5 to 0.5) of a voxel in a leaf node, and the relative position (normalized to -0.5 to 0.5) of a leaf node in the topology tree. The sum of the above three parts contains both the long-wave length and the short-wave length error.

In our experiments, we find that only the multi-color Gauss-Seidel smoother provides stable convergence when solving the linear system. For the baseline linear system solver, we choose DPCG and ICPCG (Batty). DPCG is used in Houdini for solving the viscosity equation on a uniform and adaptive grid [Goldade et al. 2019; Side Effects Software 2021] and the ICPCG (Batty) is the best baseline for the Poisson equation. We observe that DPCG is more efficient in

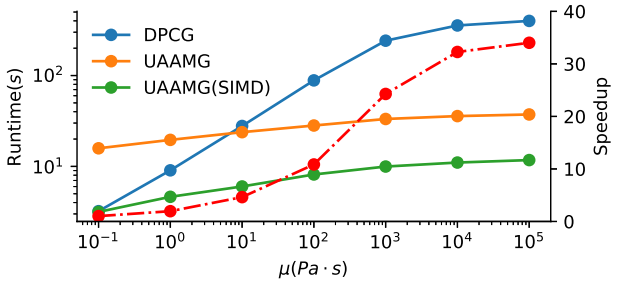


Fig. 9. Comparison of runtime (refers to the left vertical axis) with regard to different viscosity coefficients for three methods, DPCG, UAAMG and UAAMG (SIMD). The red dash dotted line is the speedup (refers to the right vertical axis) of UAAMG (SIMD) over DPCG.

solving the viscosity equation. The comparison is done with two different dynamic viscosity coefficients under different resolutions. As shown in Table 3, in both low and high viscosity tests, our approach with SIMD-VDB implementation achieves the best performance. Our approach achieves more speedup in a higher resolution scene. Using a 512 resolution, our framework can achieve 4× and 64× speedup over DPCG for viscosities of $\mu = 1$ Pa · s and $\mu = 10^4$ Pa · s respectively. Like in case of the pressure Poisson equation, we also report the performance for NT (No Trim). This technique alone achieves a speedup of 1.45× and 1.49× for two different viscosities. Moreover, in the work of Aanjaneya et al. [2019], the geometric multigrid (119.6 s) for the 3D benchmark (256^3) is slower than ICPCG (95.3 s), but ours is at least 10× faster than ICPCG. Using the same baseline method, UAAMG is proved to be much more efficient than GMG in solving the variational viscosity equation. We also tried AMGCL, however it fails unless inefficient serial Gauss-Seidel is used.

With the observation that the performance of solving the variational viscosity Eq. (8) is dependent on the viscosity coefficient, we further analyze this under the resolution of 256 with different dynamic viscosity coefficients ranging from $\mu = 0.1$ Pa · s to $\mu = 10^5$ Pa · s. As shown in Figure 9, both UAAMG and DPCG require longer time when solving a stiffer viscosity equation, but our approach is much less sensitive to material stiffness. Please note, that the numerical stiffness of Eq. (8) actually depends on $\frac{\mu \Delta t}{\rho \Delta x^2}$. Our framework is advantageous especially when high resolution, large time steps, and high viscosity are required.

7 COMPLEX NUMERICAL EXAMPLES

In this section, we compare the performance of our framework with Houdini's [Side Effects Software 2021] fluid solver in which adaptive pressure solver and state-of-the-art adaptive variational viscosity solver are implemented. In our framework, we choose the Scheduled Jacobian smoother (SRJ) for the pressure Poisson equation and the multi-color Gauss-Seidel smoother for the variational viscosity equation. We enable the adaptive solver for the pressure Poisson equation [Losasso et al. 2004] and adaptive variational viscosity equation [Goldade et al. 2019] in Houdini's setting. Same minimum and maximum FLIP substep number per video frame and same CFL

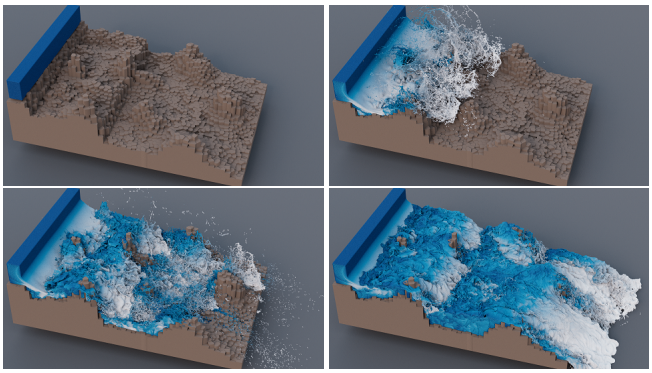


Fig. 10. **River Fall.** Water flowing down the river while hitting the riverbed. The domain size is about 75 m × 50 m × 25 m, and the grid resolution is 0.1 m.

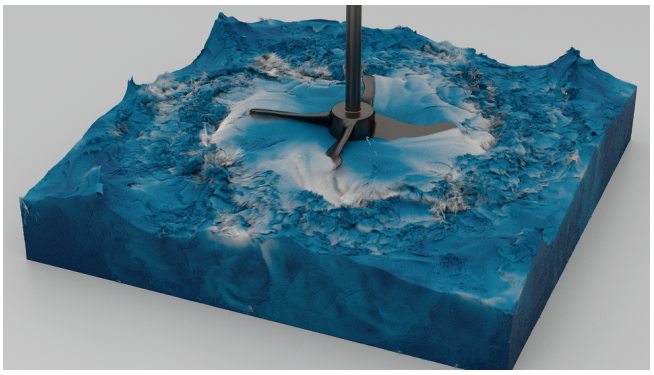


Fig. 11. **Fan Mixer.** A fan rotating in a tank of water. The turbine blades interact with the water and generate splashes and waves. The domain size is about 3 m × 3 m × 0.5 m, and the grid resolution is 0.0033 m.

number limits are set for both approaches. The numbers of total substeps are reported to give a fair comparison by calculating per-substep performance. According to Houdini's default settings, for our method we also set the relative error tolerance to be 10^{-4} for the pressure Poisson equation and 10^{-3} for the variational viscosity equation. The speedup achieved by our framework is presented in both non-viscous and viscous fluid simulations.

7.1 Non-viscous Fluids

In the simulation of non-viscous fluids, the pressure Poisson Eq. (4) is solved once in each substep. In Table 4, we report the total pressure Poisson equation solving time, total simulation time as well as their per-substep statistics. As mentioned before, we use the per-substep speedup as the main indicator. In this regard, we achieve up to 4.19× speedup for the pressure Poisson equation and 5.32× speedup for the total simulation.

7.1.1 River Fall. The river fall scene is designed as a realistic example with both compact water flow and sparse water droplets. As shown in Figure 10, water is flushing downstream. Although a lot of splashes are generated when the fluid hits the rocks, the majority of the water volume still sits on the riverbed. Our framework efficiently simulates this scene and achieves an overall speedup of 3.55× over adaptive solvers in Houdini.

7.1.2 Fan Mixer. In this non-viscous fan mixer scene, almost a billion particles are used for the detailed simulation. As shown in Figure 11, the rotating turbine blades interact with the liquid. Different levels of details, long waves and small-scale splashes, are well captured in our simulation. In Houdini's adaptive fluid solver, the simulation crashed after 40 frames due to insufficient memory. Using the statistics of this range, we achieves an overall speedup of 2.94×.

7.1.3 Meteor. In Figure 12, the dynamics of a meteor hitting into a tank of water is presented. Detailed large-scale splashes are generated after the meteor collides with the water surface. In the settling down process of the meteor, the water waves of different levels are also captured. Almost a billion particles are used in this scene.

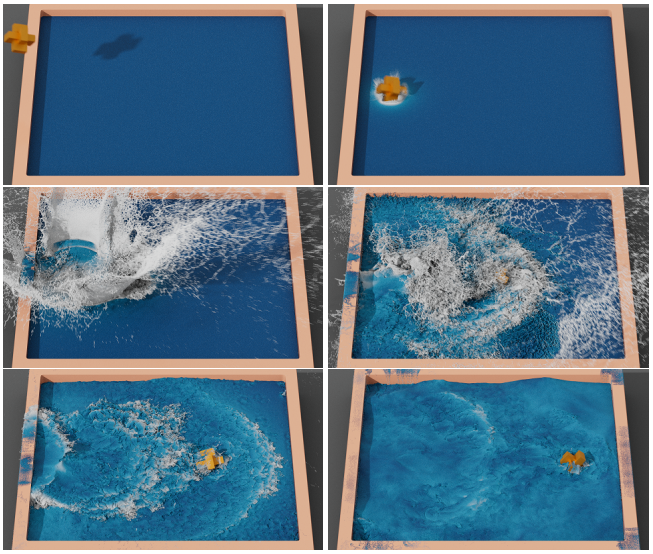


Fig. 12. **Meteor.** A meteor hitting into a tank of water and generating large-scale splashes. In the setting down process of the meteor, water waves are propagating back and forth inside the tank. The domain size is about 30 m × 30 m × 2 m, and the grid resolution is 0.025 m.

In Houdini's adaptive fluid solver, the simulation crashed after 60 frames due to insufficient memory. Compared to the octree approach [Losasso et al. 2004] from Houdini, our framework deals better with the large-scale splashes, and achieves an overall speedup of 5.32× in the corresponding frame range.

7.2 Viscous Fluids

For the viscous fluids, we further report the variational viscosity equation solving time as well as its substep statistics besides what we reported for non-viscous fluids. Depending on the different dynamic behavior and viscosity coefficients, we achieve a speedup of 2.11× to 2.91× for the pressure Poisson equation, 1.14× to 18.34× for the

Table 4. Time breakdown of the non-viscous simulations for our approach (top part in each row) and the adaptive octree approach (bottom part in each row) by running Houdini [Side Effects Software 2021]. The corresponding per-substep results are reported in the round parenthesis. Please note that the second value of per-substep runtime in our approach is calculated using statistics from the same frame range of the corresponding Houdini simulation.

Scene	Particles Count	Frames Count	Substeps Count	Pressure Time	Pressure Speedup	Total Time	Total Speedup
River Fall	165M (final)	720	2,836 2,865	6,159s (2.15s) 19,519s (6.81s)	3.17× (3.17×)	25,009s (8.74s) 88,900s (31.03s)	3.55× (3.55×)
Fan Mixer	974M	240 40 (crashed)	3,599 571	43,960s (12.21s, 12.81s) 16,197s (28.37s)	N/A (2.21×)	215,683s (59.83s, 61.47s) 103,140s (180.63s)	N/A (2.94×)
Meteor	908M	360 60 (crashed)	3,439 430	38,334s (11.15s, 13.01s) 23,428s (54.48s)	N/A (4.19×)	206,857s (60.15s, 63.98s) 146,400s (340.47s)	N/A (5.32×)

Table 5. Time breakdown of the viscous simulations for our approach (top part in each row) and the adaptive octree approach (bottom part in each row) by running Houdini [Side Effects Software 2021]. The corresponding per-substep results are reported in the round parenthesis.

Scene	Particles Count	Frames Count	Substeps Count	Pressure Time	Pressure Speedup	Viscosity Time	Viscosity Speedup	Total Time	Total Speedup
SIGGRAPH Bunnies	49M	360	386 390	1,156s (2.99s) 2,899s (7.43s)	2.51× (2.48×)	3,805s (9.86s) 11,152s (28.60s)	2.93× (2.90×)	6,253s (16.20s) 26,513s (67.98s)	4.24× (4.20×)
Bunny Cut	93M	480	2,486 1,386	7,798s (3.14s) 10,147s (7.32s)	1.30× (2.33×)	28,606s (11.51s) 143,276s (103.37s)	5.01× (8.98×)	48,758s (19.61s) 182,878s (131.95s)	3.75× (6.73×)
Fan Mixer	288M	240	1,663 2,214	14,296s (8.60s) 52,007s (23.49s)	3.64× (2.73×)	46,436s (27.92s) 70,444s (31.82s)	1.52× (1.14×)	85,618s (51.48s) 228,120s (103.04s)	2.66× (2.00×)
Meteor	114M	360	1,181 3,439	4,268s (3.61s) 34,970s (10.17s)	8.19× (2.81×)	12,459s (10.55s) 52,176s (15.17s)	4.19× (1.44×)	24,411s (20.67s) 191,544s (55.70s)	7.85× (2.69×)
Buckling/Coiling $\mu = 50$ Pa · s	34M (Final)	600	5,966 5,975	3,949s (0.66s) 8,418s (1.41s)	2.13× (2.13×)	15,571s (2.61s) 34,807s (5.83s)	2.24× (2.23×)	26,147s (4.38s) 64,080s (10.72s)	2.45× (2.45×)
Buckling/Coiling $\mu = 500$ Pa · s	41M (Final)	600	5,940 5,975	4,561s (0.77s) 9,662s (1.62s)	2.12× (2.11×)	19,511s (3.28s) 85,522s (14.31s)	4.38× (4.36×)	31,560s (5.31s) 118,500s (19.83s)	3.75× (3.73×)
Buckling/Coiling $\mu = 5\,000$ Pa · s	39M (Final)	600	5,821 5,981	5,310s (0.91s) 15,875s (2.65s)	2.99× (2.91×)	24,031s (4.13s) 452,880s (75.72s)	18.85× (18.34×)	36,920s (6.34s) 554,760s (92.75s)	15.03× (14.62×)

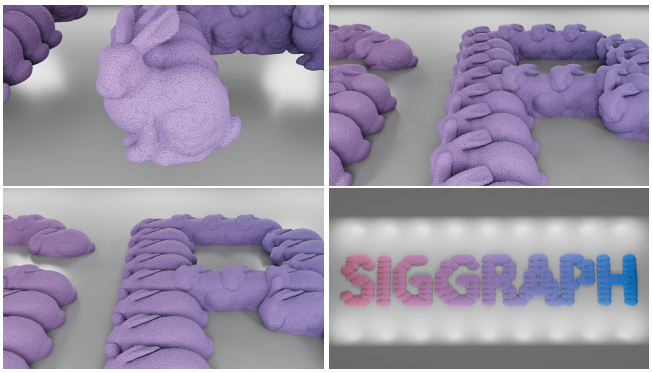


Fig. 13. **SIGGRAPH Bunnies.** Hundreds of viscous bunnies are dropped at the same time resulting in a *SIGGRAPH*-lettering. The dynamic viscosity coefficient is $\mu = 5\,000$ Pa · s. The height of a single bunny is about 1.15 m and the grid resolution is 0.02 m.

variational viscosity equation, and 2.00× to 14.62× for the total simulation using per-substep statistics. Table 5 shows the summary.

7.2.1 SIGGRAPH Bunnies. In Figure 13, hundreds of viscous bunnies are released at the same time resulting in a *SIGGRAPH*-lettering. In the initialization of FLIP, one voxel on average contains 8 particles. In this scene, the DOFs count for the pressure Poisson equation (as well as voxel count) is 7 million and the variational viscosity equation is 23 million. This is already a challenging scene while our framework can finish the simulation in a very short time (less than 2 hours), achieving 2.48× and 2.90× speedup for the pressure Poisson equation and variational viscosity equation respectively. Please note, that the larger overall speedup of 4.20× is achieved by optimization using the SIMD for particle advection and P2G/G2P transformation.

7.2.2 Bunny Cut. Our framework can efficiently simulate the challenging scene reported in Goldade et al. [2019] in which a highly viscous bunny interacts with two crossed thin wires. As shown in Figure 14, the bunny is cut by the wires after being dropped onto them. It then deforms and coalesces after the cut. This scene is highly viscous ($\mu = 20\,000$ Pa · s) and is simulated using a high resolution. As analyzed in the unit test experiments section, our framework performs particularly well in such scenarios and achieves a per-substep speedup of 8.98× for solving the variational viscosity equation compared to Houdini's adaptive octree approach [Goldade et al. 2019].

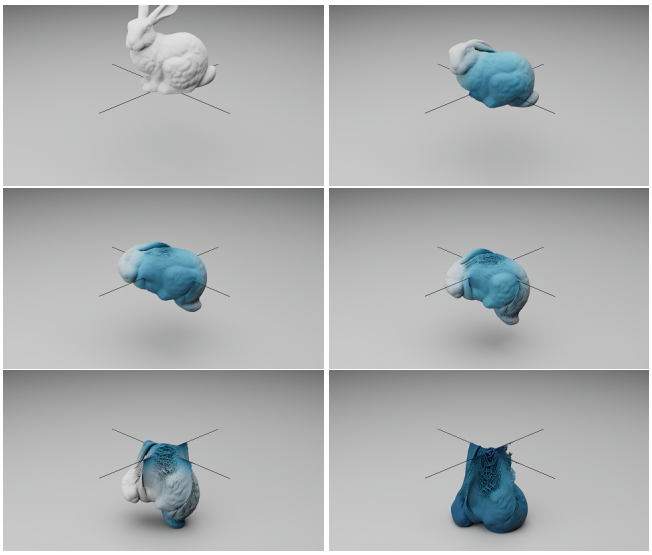


Fig. 14. **Bunny Cut.** A highly viscous bunny hitting thin wires. The bunny deforms and coalesces after the cut. The dynamic viscosity coefficient is $\mu = 20\,000$ Pa · s. The height of the bunny is about 1.57 m and the grid resolution is 0.004 m.

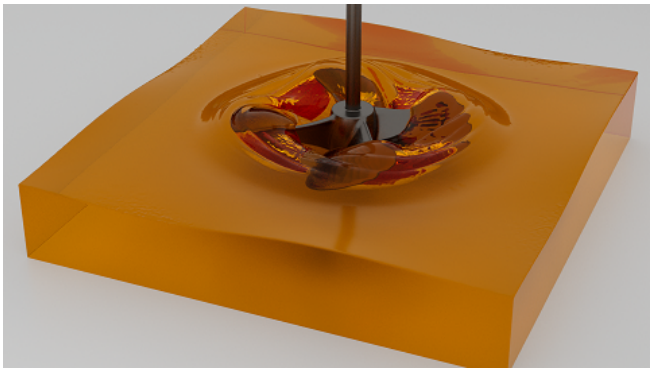


Fig. 15. **Fan Mixer (Viscous).** A fan rotating in a tank of viscous liquid. Long-length waves are generated along the box. The dynamic viscosity coefficient is $\mu = 2\,000$ Pa · s. The domain size is about 3 m × 3 m × 0.5 m and the grid resolution is 0.005 m.

7.2.3 Fan Mixer. In Figure 15, the fan mixer's blades interact with a viscous liquid. The dynamics is less intense compared to the non-viscous scene. Most volume of the liquid almost remains static, thus the solving of the variational viscosity equation is relatively not challenging for the Houdini adaptive octree approach. Both approaches perform well. Per-substep speedups of 1.14× and 2.00× are achieved for the variational viscosity equation and the overall simulation respectively.

7.2.4 Meteor. In the viscous scene of a meteor hitting into a tank of liquid, the large-scale splashes are not generated in contrast to the non-viscous scene. As shown in Figure 16, the meteor grooves a trajectory in the viscous liquid which is then smoothed gradually.

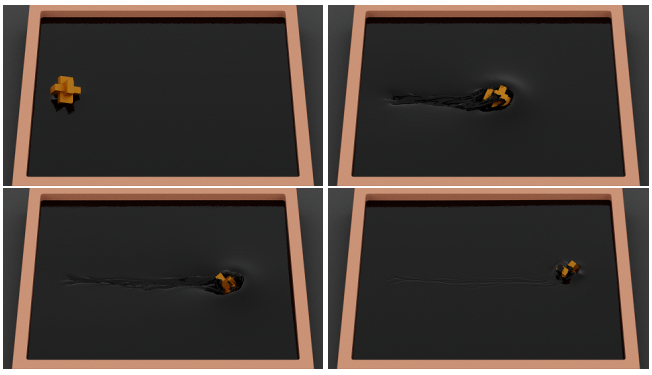


Fig. 16. **Meteor (Viscous).** A meteor hitting into a tank of viscous liquid. The grooved trajectory by the meteor is gradually smoothed. The dynamic viscosity coefficient is $\mu = 5\,000$ Pa · s. The domain size is about 30 m × 30 m × 2 m and the grid resolution is 0.05 m.

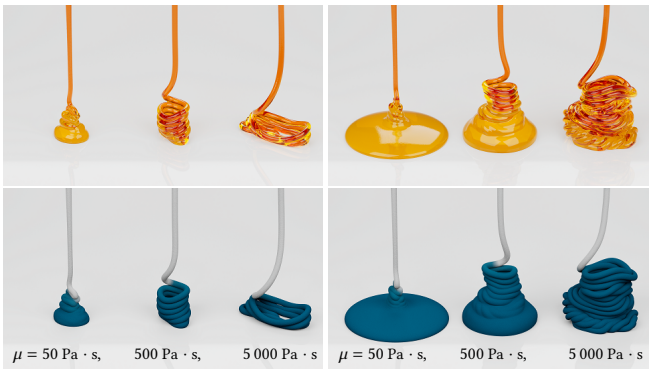


Fig. 17. **Buckling/Coiling.** Buckling and coiling effects for various viscosity coefficients. Mesh (top) and particle views (bottom) are shown at different points in time (left to right). The viscous thread has a radius of 0.005 m and the grid resolution is 0.0005 m.

Both approaches perform well in solving the variational viscosity equation and our framework achieves a per-substep speedup of 1.44× for solving this equation and an overall speedup of 2.69×.

7.2.5 Buckling/Coiling. Buckling and coiling phenomena are commonly observed in our daily life when a jet of viscous liquid hits onto a ground. Our framework can easily simulate the corresponding dynamics with various viscosity coefficients as shown in Figure 17. In the unit test experiments section for the viscosity equation, we already showcased that our framework is less sensitive to numerical stiffness. We achieve a larger speedup compared to the adaptive octree approach. In the most viscous scene, we achieve a 18.34× speedup for solving the variational viscosity equation. Furthermore, we observed that Houdini's adaptive viscosity solver actually reaches its iteration limit and does not satisfy the convergence criterion of a relative error of 10^{-3} for $\mu = 5\,000$ Pa · s. Our framework is not only faster but also more stable in such a challenging scene.

8 CONCLUSION, LIMITATIONS, AND FUTURE WORK

In this work, we have demonstrated that our Unsmoothed Aggregation Algebraic MultiGrid (UAAMG) framework can achieve state-of-the-art performance on the simulation of incompressible flow. Compared to the newest version of Houdini 19.0 [Side Effects Software 2021], our framework is about 2.11× to 4.19× faster than the adaptive pressure solver [Losasso et al. 2005, 2004], 1.14× to 18.34× faster than the adaptive variational viscosity solver [Goldade et al. 2019], and 2.00× to 14.62× faster overall.
The following four key components contribute to our success:

(1) a SIMD accelerated matrix-vector multiplication shortens the time for each iteration;
(2) a matrix coefficients trimming strategy improves computation throughput in the matrix-vector multiplication;
(3) an unsmoothed aggregation multigrid preconditioned conjugate gradient method converges in a few iterations over a wide range of parameters;
(4) a parallel coarsening implementation ensures short construction time for the different multigrid levels.

While each component substantially improves the performance, implementing all components requires considerable effort. Finally, we give some practical advice based on the amount of effort, and availability of source codes.

Without SIMD and parallel coarsening (basic implementation), the naive UAAMG based on Eigen performs comparable to AMGCL for the Poisson equation and slower than the adaptive pressure solver in Houdini. For the variational viscosity equation, naive UAAMG takes about 4× longer, slower than the adaptive method [Goldade et al. 2019] in Houdini. Therefore we recommend Houdini in general, while AMGCL is the second option for the Poisson equation. AMGCL fails in solving the variational viscosity equation.

When SIMD, trimming, and matrix-free coarsening are incorporated (full implementation), our framework is recommended over all large scale examples, even in the very splashy river fall scene (see Figure 10) where many SIMD vectors are only sparsely filled with few DOFs.

Our Poisson solver is a general solver using a symmetric 7-point stencil on regular Cartesian grids. It only requires the matrix coefficients on the finest grid and the spatial pattern of the DOFs. It shows great potential beyond the pressure projection in fluid simulation. In theory, our solver can be used for Poisson surface reconstruction [Kazhdan et al. 2006], and solving the magnetic field in an Eulerian ferrofluid simulation [Ni et al. 2020].

Both of the solvers rely on regular grids, which is not memory efficient compared to adaptive grids. The adaptive method of Ando and Batty [2020] could have a higher speedup than ours, but they have compromised accuracy. The speedup of SIMD relies on high volume portions of DOFs inside SIMD vectors resulting in a reduced acceleration in very splashy scenes. When the number of particles is small, the overhead is too large. The currently used multi-color Gauss-Seidel smoother works well in the viscosity solver, but the iteration number still weakly depends on the viscosity, while Zhu et al. [2010] showed a smoother which is insensitive to material parameters. It would be interesting to see if such a smoother can be incorporated. In the presence of spatially varying viscosity coefficients, there are not much matrix coefficients to trim off. Therefore the performance would be affected in such a case.

Moreover, several future improvements could be promising. The current solver uses 60 GB/s of memory bandwidth in the solving process, which is almost the peak bandwidth for four channel DDR4-2133 memory. Moving the whole pipeline to the GPU with large memory bandwidth should be explored. The current code only runs on a single workstation, it is conceivable to extend the system to multiple compute nodes.

ACKNOWLEDGMENTS

This work has been supported and funded by KAUST through individual baseline funding. The authors are grateful to Xinxin Zhang for providing the code of the first version of the unsmoothed aggregation-based multigrid Poisson solver. The anonymous reviewers' valuable comments that improved the manuscript are gratefully acknowledged.

REFERENCES

Mridul Aanjaneya, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. 2017. Power Diagrams and Sparse Paged Grids for High Resolution Adaptive Liquids. *ACM Trans. Graph.* 36, 4, Article 140 (2017), 12 pages.

Mridul Aanjaneya, Chengguizi Han, Ryan Goldade, and Christopher Batty. 2019. An Efficient Geometric Multigrid Solver for Viscous Liquids. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 2, 2 (2019), 1–21.

Ryoichi Ando and Christopher Batty. 2020. A Practical Octree Liquid Simulator with Adaptive Surface Resolution. 39, 4, Article 32 (2020), 17 pages.

Christopher Batty. 2018. VariationalViscosity3D. https://github.com/christopherbatty/VariationalViscosity3D.

Christopher Batty, Florence Bertails, and Robert Bridson. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 100–es.

Christopher Batty and Robert Bridson. 2008. Accurate viscous free surfaces for buckling, coiling, and rotating liquids. (2008).

Jan Bender and Dan Koschier. 2017. Divergence-free SPH for incompressible and viscous fluids. *IEEE Transactions on Visualization and Computer Graphics* 23, 3 (March 2017), 1193–1206.

Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. 2003. Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid. *ACM Trans. Graph.* 22, 3 (July 2003), 917–924.

Robert Bridson. 2016. *Fluid Simulation for Computer Graphics.* CRC Press.

Robert Bridson and Matthias Müller-Fischer. 2007. Fluid simulation: SIGGRAPH 2007 course notes Video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 courses.* 1–81.

William L. Briggs, Van Emden Henson, and Steve F. McCormick. 2000. *A multigrid tutorial.* SIAM.

Oliver Bröker, Marcus J. Grote, Carsten Mayer, and Arnold Reusken. 2001. Robust Parallel Smoothing for Multigrid Via Sparse Approximate Inverses. 23, 4 (apr 2001), 1396–1417.

Nuttapong Chentanez and Matthias Müller. 2011. Real-time Eulerian water simulation using a restricted tall cell grid. *ACM Trans. Graph.* 30, 4 (2011), 82.

Nuttapong Chentanez and Matthias Müller-Fischer. 2012. A Multigrid Fluid Pressure Solver Handling Separating Solid Boundary Conditions. *IEEE Transactions on Visualization and Computer Graphics* 18, 8 (2012), 1191–1201.

Alexandre Joel Chorin. 1967. The numerical solution of the Navier-Stokes equations for an incompressible fluid. *Bull. Amer. Math. Soc.* 73, 6 (1967), 928–931.

Denis Demidov. 2019. AMGCL: An Efficient, Flexible, and Extensible Algebraic Multigrid Implementation. *Lobachevskii Journal of Mathematics* 40, 5 (01 May 2019), 535–546.

Denis Demidov. 2020. AMGCL – A C++ library for efficient solution of large sparse linear systems. *Software Impacts* 6 (2020), 100037.

Christian Dick, Marcus Rogowsky, and Rüdiger Westermann. 2015. Solving the fluid pressure Poisson equation using multigrid—evaluation and improvements. *IEEE transactions on visualization and computer graphics* 22, 11 (2015), 2480–2492.

Douglas Enright, Stephen Marschner, and Ronald Fedkiw. 2002. Animation and Rendering of Complex Water Surfaces. *ACM Trans. Graph.* 21, 3 (July 2002), 736–744.

Florian Ferstl, Rüdiger Westermann, and Christian Dick. 2014. Large-Scale Liquid Simulation on Adaptive Hexahedral Grids. *IEEE Transactions on Visualization and Computer Graphics* 20, 10 (2014), 1405–1417.

Nick Foster and Ronald Fedkiw. 2001. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 23–30.
Frederic Gibou, Ronald P Fedkiw, Li-Tien Cheng, and Myungjoo Kang. 2002. A second-order-accurate symmetric discretization of the Poisson equation on irregular domains. *J. Comput. Phys.* 176, 1 (2002), 205–227.
Ryan Goldade, Yipeng Wang, Mridul Aanjaneya, and Christopher Batty. 2019. An Adaptive Variational Finite Difference Framework for Efficient Symmetric Octree Viscosity. *ACM Trans. Graph.* 38, 4, Article 94 (July 2019), 14 pages.
Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.
Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. 2014a. Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics* 20, 3 (March 2014), 426–435.
Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. 2014b. SPH Fluids in Computer Graphics. In *Eurographics 2014 - State of the Art Reports*.
Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The Affine Particle-in-Cell Method. *ACM Trans. Graph.* 34, 4, Article 51 (July 2015), 10 pages.
Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson Surface Reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (Cagliari, Sardinia, Italy) *(SGP '06)*. Eurographics Association, Goslar, DEU, 61–70.
Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. 2019. Smoothed Particle Hydrodynamics for Physically-Based Simulation of Fluids and Solids. In *EUROGRAPHICS 2019 Tutorials*. The Eurographics Association.
Egor Larionov, Christopher Batty, and Robert Bridson. 2017. Variational Stokes: A Unified Pressure-Viscosity Solver for Accurate Viscous Liquids. *ACM Trans. Graph.* 36, 4, Article 101 (July 2017), 11 pages.
Haixiang Liu, Yuanming Hu, Bo Zhu, Wojciech Matusik, and Eftychios Sifakis. 2018. Narrow-Band Topology Optimization on a Sparsely Populated Grid. *ACM Trans. Graph.* 37, 6, Article 251 (dec 2018), 14 pages.
Frank Losasso, Ronald Fedkiw, and Stanley Osher. 2005. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids* 35 (2005), 2006.
Frank Losasso, Frédéric Gibou, and Ron Fedkiw. 2004. Simulating Water and Smoke with an Octree Data Structure. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 457–462.
Chaoyang Lyu, Wei Li, Mathieu Desbrun, and Xiaopei Liu. 2021. Fast and Versatile Fluid-Solid Coupling for Turbulent Flow Simulation. *ACM Trans. Graph.* 40, 6, Article 201 (dec 2021), 18 pages.
Aleka McAdams, Eftychios Sifakis, and Joseph Teran. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Madrid, Spain) *(SCA '10)*. Eurographics Association, Goslar, DEU, 65–74.
Nathan Mitchell, Michael Doescher, and Eftychios Sifakis. 2016. A Macroblock Optimization for Grid-Based Nonlinear Elasticity. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Zurich, Switzerland) *(SCA '16)*. Eurographics Association, Goslar, DEU, 11–19.
Jeroen Molemaker, Jonathan M. Cohen, Sanjit Patel, and Jonyong Noh. 2008. Low Viscosity Flow Simulations for Animation *(SCA '08)*. Eurographics Association, Goslar, DEU, 9–18.
Matthias Müller-Fischer, David Charypar, and Markus Gross. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '03)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 154–159.
Ken Museth. 2013. VDB: High-Resolution Sparse Volumes with Dynamic Topology. *ACM Trans. Graph.* 32, 3, Article 27 (June 2013), 22 pages.
Yen Ting Ng, Chohong Min, and Frédéric Gibou. 2009. An efficient fluid–solid coupling algorithm for single-phase flows. *J. Comput. Phys.* 228, 23 (2009), 8807–8829.
Xingyu Ni, Bo Zhu, Bin Wang, and Baoquan Chen. 2020. A Level-Set Method for Magnetic Substance Simulation. *ACM Trans. Graph.* 39, 4, Article 29 (July 2020), 15 pages.
John W Ruge and Klaus Stüben. 1987. Algebraic multigrid. In *Multigrid methods*. SIAM, 73–130.
Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: A Sparse Paged Grid Structure Applied to Adaptive Smoke Simulation. *ACM Trans. Graph.* 33, 6, Article 205 (Nov. 2014), 12 pages.
Side Effects Software. 2021. Houdini 19.0. (2021).
Barbara Solenthaler and Renato Pajarola. 2009. Predictive-corrective incompressible SPH. *ACM Trans. Graph.* 28, 3, Article 40 (2009), 6 pages.
Jos Stam. 1999. Stable Fluids. *Proc. of ACM SIGGRAPH* (1999), 121–128.
Klaus Stüben. 2001. A review of algebraic multigrid. *J. Comput. Appl. Math.* 128, 1 (2001), 281–309. Numerical Analysis 2000. Vol. VII: Partial Differential Equations.
Tetsuya Takahashi and Ming C. Lin. 2019. A Geometrically Consistent Viscous Fluid Solver with Two-Way Fluid-Solid Coupling. *Computer Graphics Forum* 38, 2 (2019), 49–58.
Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed Aggregation Multigrid for Cloth Simulation. *ACM Trans. Graph.* 34, 6, Article 245 (oct 2015), 13 pages.
Nils Thürey and Ulrich Rüde. 2009. Stable Free Surface Flows with the Lattice Boltzmann Method on Adaptively Coarsened Grids. *Comput. Vis. Sci.* 12, 5 (June 2009), 247–263.
Peter Vaněk, Marian Brezina, and Jan Mandel. 2001. Convergence of algebraic multigrid based on smoothed aggregation. *Numer. Math.* 88, 3 (2001), 559–579.
Petr Vaněk, Jan Mandel, and Marian Brezina. 1996. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing* 56, 3 (1996), 179–196.
Xinlei Wang, Minchen Li, Yu Fang, Xinxin Zhang, Ming Gao, Min Tang, Danny M. Kaufman, and Chenfanfu Jiang. 2020. Hierarchical Optimization Time Integration for CFL-Rate MPM Stepping. *ACM Trans. Graph.* 39, 3, Article 21 (Apr 2020), 16 pages.
Zhendong Wang, Longhua Wu, Marco Fratarcangeli, Min Tang, and Huamin Wang. 2018. Parallel multigrid for nonlinear cloth simulation. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 131–141.
Daniel Weber, Johannes Mueller-Roemer, André Stork, and Dieter Fellner. 2015. A Cut-Cell Geometric Multigrid Poisson Solver for Fluid Simulation. *Computer Graphics Forum* 34, 2 (2015), 481–491.
Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A Scalable Galerkin Multigrid Method for Real-Time Simulation of Deformable Objects. *ACM Trans. Graph.* 38, 6, Article 162 (nov 2019), 13 pages.
Xiang Yang and Rajat Mittal. 2017. Efficient relaxed-Jacobi smoothers for multigrid on parallel computers. *J. Comput. Phys.* 332 (2017), 135–142.
Omar Zarifi. 2020. Sparse Smoke Simulations in Houdini. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks*. 1–2.
Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 965–972.
Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An Efficient Multigrid Method for the Simulation of High-Resolution Elastic Solids. *ACM Trans. Graph.* 29, 2, Article 16 (apr 2010), 18 pages.

Appendix 1 EXPANDED VARIATIONAL VISCOSITY EQUATION DISCRETIZATION

In a staggered grid, the cell with index (i, j, k) occupies the space of the unit cube

$$\left[i-\frac{1}{2}, i+\frac{1}{2}\right] \times \left[j-\frac{1}{2}, j+\frac{1}{2}\right] \times \left[k-\frac{1}{2}, k+\frac{1}{2}\right].$$

Velocity components in three directions $u_{i,j,k}$, $v_{i,j,k}$, $w_{i,j,k}$ and corresponding volume fractions are defined at face centers $(i-\frac{1}{2}, j, k)$, $(i, j-\frac{1}{2}, k)$, $(i, j, k-\frac{1}{2})$ respectively. Pressure volume fraction $V_{p;i,j,k}$ is defined at cell center (i, j, k). Volume fractions of stress tensor components $V_{\tau_{xy};i,j,k}$, $V_{\tau_{yz};i,j,k}$, $V_{\tau_{xz};i,j,k}$ are defined at edge centers $(i-\frac{1}{2}, j-\frac{1}{2}, k)$, $(i, j-\frac{1}{2}, k-\frac{1}{2})$, $(i-\frac{1}{2}, j, k-\frac{1}{2})$ respectively.

Figure A1 shows the cross section of the velocity component u's stencil at index (i, j, k) on the plane $z = k$ and $y = j$. The six off-diagonal terms are $u_{i-1,j,k}$, $u_{i+1,j,k}$, $u_{i,j-1,k}$, $u_{i,j+1,k}$, $u_{i,j,k-1}$, $u_{i,j,k+1}$. The eight cross terms are $v_{i,j,k}$, $v_{i,j+1,k}$, $v_{i-1,j,k}$, $v_{i-1,j+1,k}$, $w_{i,j,k}$, $w_{i,j,k+1}$, $w_{i-1,j,k}$, $w_{i-1,j,k+1}$.

The discretization of variational viscosity equation at $u_{i,j,k}$ is given by

$$\begin{aligned}
&\left(V_{u;i,j,k} + \frac{\mu\Delta t}{\rho\Delta x^2}\left(2V_{p;i,j,k} + V_{\tau_{xy};i,j,k} + V_{\tau_{xz};i,j,k} + \right.\right.\\
&\qquad \left.\left. 2V_{p;i-1,j,k} + V_{\tau_{xy};i,j+1,k} + V_{\tau_{xz};i,j,k+1}\right)\right) u_{i,j,k}\\
&\quad - \frac{2\mu\Delta t V_{p;i-1,j,k}}{\rho\Delta x^2} u_{i-1,j,k} - \frac{2\mu\Delta t V_{p;i,j,k}}{\rho\Delta x^2} u_{i+1,j,k}\\
&\quad - \frac{\mu\Delta t V_{\tau_{xy};i,j,k}}{\rho\Delta x^2} u_{i,j-1,k} - \frac{\mu\Delta t V_{\tau_{xy};i,j+1,k}}{\rho\Delta x^2} u_{i,j+1,k}\\
&\quad - \frac{\mu\Delta t V_{\tau_{xz};i,j,k}}{\rho\Delta x^2} u_{i,j,k-1} - \frac{\mu\Delta t V_{\tau_{xz};i,j,k+1}}{\rho\Delta x^2} u_{i,j,k+1}
\end{aligned}$$

$$
\begin{aligned}
&+\frac{\mu\Delta t V_{\tau_{xy};i,j,k}}{\rho\Delta x^2}v_{i,j,k} - \frac{\mu\Delta t V_{\tau_{xy};i,j+1,k}}{\rho\Delta x^2}v_{i,j+1,k}\\
&-\frac{\mu\Delta t V_{\tau_{xy};i,j,k}}{\rho\Delta x^2}v_{i-1,j,k} + \frac{\mu\Delta t V_{\tau_{xy};i,j+1,k}}{\rho\Delta x^2}v_{i-1,j+1,k}\\
&+\frac{\mu\Delta t V_{\tau_{xz};i,j,k}}{\rho\Delta x^2}w_{i,j,k} - \frac{\mu\Delta t V_{\tau_{xz};i,j,k+1}}{\rho\Delta x^2}w_{i,j,k+1}\\
&-\frac{\mu\Delta t V_{\tau_{xz};i,j,k}}{\rho\Delta x^2}w_{i-1,j,k} + \frac{\mu\Delta t V_{\tau_{xz};i,j,k+1}}{\rho\Delta x^2}w_{i-1,j,k+1}\\
&= V_{u;i,j,k}u^{\text{old}}_{i,j,k}\,.
\end{aligned}
$$

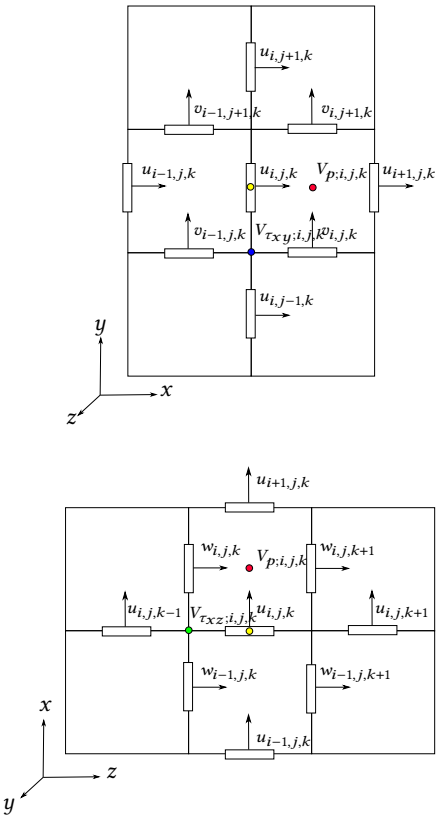


Fig. A1. Stencils for the velocity component u at index (i, j, k). There is one diagonal term, six off-diagonal terms, and eight cross terms.

Appendix 2 THRESHOLD VOLUME FRACTION

As mentioned in Section 3.2, the l.h.s. matrix of Eq. (8) is singular in some scenarios. In Figure A2, we show that the singularity is related to the cross component terms on the free surface.

For stress tensor components, τ_{xx} and τ_{yy} centered at the red circle, we obtain a volume fraction of $V_p = 0.25$. τ_{xy} between velocity components u_0 and u_3 also has a volume fraction of $V_{\tau_{xy}} = 0.25$. All other stress tensor components have a volume fraction $V = 0$. Velocity components u_0 and u_3 have the volume fraction $V = 0.25$, and u_1, u_2, u_4, u_5 have the volume fraction $V = 0$. In our implementation, a threshold volume fraction with $V = \epsilon$ is set for all velocity components. The variational viscosity equation in this scenario is expanded as

$$
\begin{aligned}
u_0 - \frac{\mu\Delta t}{\rho\Delta x^2}\left(2(u_2-u_0)+(u_4-u_0)+(u_3-u_5)\right) &= u_0^{\text{old}}\,,\\
4\epsilon u_1 - \frac{\mu\Delta t}{\rho\Delta x^2}\left(2(u_3-u_1)\right) &= 4\epsilon u_1^{\text{old}}\,,\\
4\epsilon u_2 - \frac{\mu\Delta t}{\rho\Delta x^2}\left(-2(u_2-u_0)\right) &= 4\epsilon u_2^{\text{old}}\,,\\
u_3 - \frac{\mu\Delta t}{\rho\Delta x^2}\left(-2(u_3-u_1)-(u_4-u_0)-(u_3-u_5)\right) &= u_3^{\text{old}}\,,\\
4\epsilon u_4 - \frac{\mu\Delta t}{\rho\Delta x^2}\left(-(u_4-u_0)-(u_3-u_5)\right) &= 4\epsilon u_4^{\text{old}}\,,\\
4\epsilon u_5 - \frac{\mu\Delta t}{\rho\Delta x^2}\left((u_4-u_0)+(u_3-u_5)\right) &= 4\epsilon u_5^{\text{old}}\,.
\end{aligned}
$$

With $\alpha = \frac{\mu\Delta t}{\rho\Delta x^2}$, we obtain the equation

$$
\mathbf{H}\vec{\mathbf{u}} = \vec{\mathbf{b}}\,, \tag{A1}
$$

in which

$$
\mathbf{H} = \begin{pmatrix}
1+3\alpha & 0 & -2\alpha & -\alpha & -\alpha & \alpha\\
0 & 4\epsilon+2\alpha & 0 & -2\alpha & 0 & 0\\
-2\alpha & 0 & 4\epsilon+2\alpha & 0 & 0 & 0\\
-\alpha & -2\alpha & 0 & 1+3\alpha & \alpha & -\alpha\\
-\alpha & 0 & 0 & \alpha & 4\epsilon+\alpha & -\alpha\\
\alpha & 0 & 0 & -\alpha & -\alpha & 4\epsilon+\alpha
\end{pmatrix},
$$

$$
\vec{\mathbf{u}} = \begin{pmatrix} u_0 & u_1 & u_2 & u_3 & u_4 & u_5 \end{pmatrix}^T,
$$

$$
\vec{\mathbf{b}} = \begin{pmatrix} u_0^{\text{old}} & 4\epsilon u_1^{\text{old}} & 4\epsilon u_2^{\text{old}} & u_3^{\text{old}} & 4\epsilon u_4^{\text{old}} & 4\epsilon u_5^{\text{old}} \end{pmatrix}^T.
$$

The last two rows of the matrix $\mathbf{H}$ in Eq. (A1) would be linearly dependent for a threshold volume fraction $\epsilon = 0$ leading to singularity.

Appendix 3 CROSS TERMS WITHIN THE VARIATIONAL VISCOSITY EQUATION

Fetching the DOF SIMD vectors and matrix coefficients SIMD vectors of off-diagonal terms in the variational viscosity equation in the SIMD-VDB approach is addressed in the same way as in the case of the Poisson equation. Here, we discuss how cross terms are addressed in the SIMD-VDB approach. It depends on the velocity component channel and we will discuss them separately. Figures A1, A3, and A4 present related cross terms for the velocity component u, v and w respectively.

For the velocity component $u_{i,j,k}$, DOF SIMD vectors $v_{i,j,k}$, $v_{i-1,j,k}$, $v_{i,j+1,k}$, $v_{i-1,j+1,k}$ from the v component and DOF SIMD vectors $w_{i,j,k}$, $w_{i-1,j,k}$, $w_{i,j,k+1}$, $w_{i-1,j,k+1}$ from the w component are related cross terms as illustrated in Figure A1. The former six DOF SIMD vectors are continuous in memory since their offsets are in x and y directions. $w_{i,j,k+1}$ and $w_{i-1,j,k+1}$ require the usage of function **get positive z SIMD vector** from Figure 6 since their offsets are in z direction. The matrix coefficients SIMD vectors are continuous in memory for all eight terms.

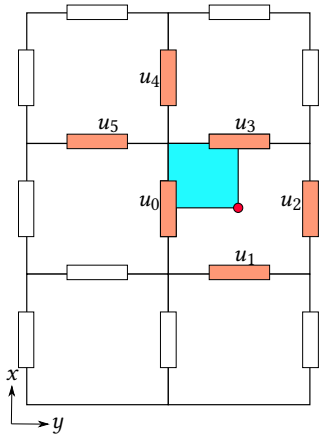


Fig. A2. Illustration of the designed 2D scenario which results in a singular matrix of the variational viscosity equation. The liquid is only occupying a quarter of one voxel as shown in the blue region. The velocity components from u_0 to u_5 sketched as orange boxes are unknowns in Eq. (8).

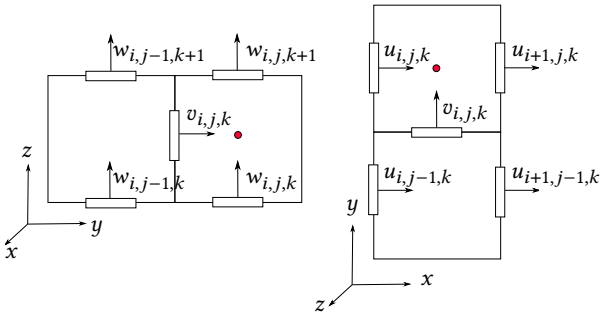


Fig. A3. Related cross terms for the velocity component v at index (i, j, k).

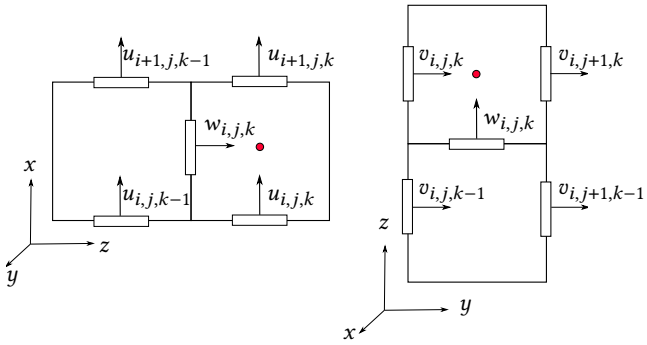


Fig. A4. Related cross terms for the velocity component w at index (i, j, k).

For the velocity component $v_{i,j,k}$, the corresponding eight cross terms are shown in Figure A3. Among the DOF SIMD vectors, $w_{i,j-1,k+1}$ and $w_{i,j,k+1}$ require the usage of function **get positive z SIMD vector** and the rest DOF SIMD vectors are continuous in memory. The matrix coefficients SIMD vectors are continuous in memory for all eight terms.

For the velocity component $w_{i,j,k}$, the corresponding eight cross terms are shown in Figure A4. Fetching DOF SIMD vectors $u_{i,j,k-1}$, $u_{i+1,j,k-1}$, $v_{i,j,k-1}$, $v_{i,j+1,k-1}$ and the corresponding matrix coefficients SIMD vectors require the usage of function **get negative z SIMD vector**. For the other terms, both DOF SIMD vectors and matrix coefficients SIMD vectors are continuous in memory.

Appendix 4 CONVERGENCE PLOT

The convergence plot for the unit test experiments is shown in Figure A5.

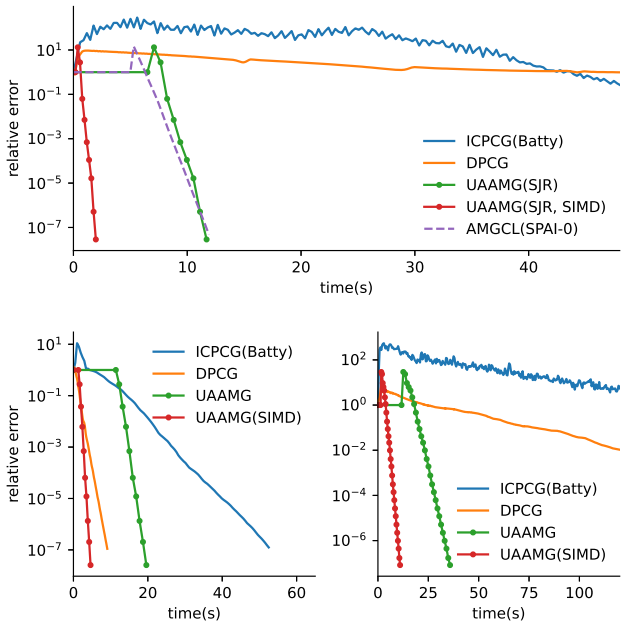


Fig. A5. Convergence plot for the unit test experiments. A comparison for the pressure Poisson equation between five methods (DPCG, ICPCG (Batty), AMGCL (SPAI-0), UAAMG (SRJ) and UAAMG (SRJ, SIMD)) using a resolution of 256 is shown at the top. A comparison for the variational viscosity equation between four methods (DPCG, ICPCG (Batty), UAAMG, UAAMG (SIMD)) using a resolution of 256 is shown at the bottom. Viscosity coefficients $\mu = 1$ Pa · s (bottom left) and $\mu = 10^4$ Pa · s (bottom right) are used. The absolute error normalized by the norm of the initial r.h.s. is shown as relative error.