A Multilevel Active-Set Preconditioner for Box-Constrained Pressure Poisson Solvers

TETSUYA TAKAHASHI, Unaffiliated, USA
CHRISTOPHER BATTY, University of Waterloo, Canada

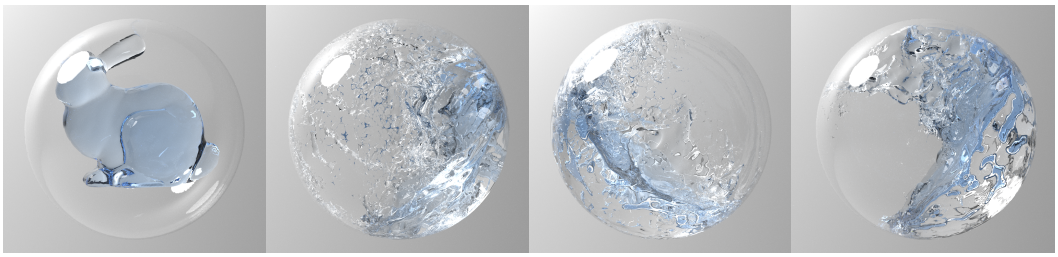

Fig. 1. Our multilevel box-constrained convex QP solver, SAAMG-MPRGP, efficiently enforces fluid incompressibility while satisfying non-negative pressure constraints for separating solid boundary conditions. Using our solver, liquid in a moving sphere is simulated without suffering from artificial suction to the solid boundary. This scene used a grid resolution of $\mathbf{192}^3$ and 1.0M particles, and on average required 29.2 seconds per frame for the entire pressure solve.

Efficiently solving large-scale box-constrained convex quadratic programs (QPs) is an important computational challenge in physical simulation. We propose a new multilevel preconditioning scheme based on the active-set method and combine it with modified proportioning with reduced gradient projections (MPRGP) to efficiently solve such QPs arising from pressure Poisson equations with non-negative pressure constraints in fluid animation. Our method employs a purely algebraic multigrid method to ensure the solvability of the coarser level systems and to merge only algebraically-connected components, thereby avoiding performance degradation of the preconditioner. We present a filtering scheme to efficiently apply our multilevel preconditioning only to unconstrained subsystems of the pressure Poisson system while reusing the hierarchy constructed per simulation step. We demonstrate the effectiveness of our method over previous approaches in various examples.

CCS Concepts: • **Computing methodologies** → **Physically-based simulation**.

Additional Key Words and Phrases: Fluid simulation, quadratic program, multigrid

ACM Reference Format:
Tetsuya Takahashi and Christopher Batty. 2023. A Multilevel Active-Set Preconditioner for Box-Constrained Pressure Poisson Solvers . *Proc. ACM Comput. Graph. Interact. Tech.* 6, 2, Article 1 (August 2023), 22 pages. https://doi.org/10.1145/3606939

Authors' addresses: Tetsuya Takahashi, Unaffiliated, USA, tetsuya@cs.unc.edu; Christopher Batty, University of Waterloo, Canada, christopher.batty@uwaterloo.ca.

2577-6193/2023/8-ART1 $15.00
https://doi.org/10.1145/3606939

1 INTRODUCTION

Enforcing fluid incompressibility while supporting *separating solid boundary conditions* in grid-based fluid simulation is essential to generate natural liquid behaviors in many practical scenarios [Batty et al. 2007; Chentanez and Müller 2012; Lai et al. 2020]. While incompressibility with standard boundary conditions can be achieved efficiently by solving Poisson problems involving only systems of linear equations [Bridson 2015], separating solid boundary conditions impose non-negative pressure constraints, i.e., inequality constraints. This modification raises computational issues because the underlying numerical task becomes a *linear complementarity problem* (LCP) or, equivalently, a box-constrained convex quadratic program (QP) [Batty et al. 2007; Narain et al. 2010]. While standalone multigrid solvers have been investigated to efficiently solve this LCP [Chentanez and Müller 2012; Lai et al. 2020], it has been observed that instead using multigrid as a preconditioner for iterative Krylov solvers (e.g., conjugate gradient (CG) [Shewchuk 1994]) can improve robustness and efficiency [McAdams et al. 2010]. As such, considering that Krylov solvers are designed to minimize quadratic objectives, we aim to efficiently solve the box-constrained convex QP via a multigrid-preconditioned Krylov solver.

In this paper, we propose a new multilevel active-set preconditioning scheme and combine it with an active-set Krylov solver, specifically *modified proportioning with reduced gradient projections* (MPRGP) [Dostal and Schoberl 2005; Dostl 2009], to efficiently solve box-constrained symmetric positive-definite (SPD) QPs. In particular, we focus on the large-scale sparse QP which arises from pressure projection with non-negative pressure constraints to enforce fluid incompressibility with separating solid boundary conditions. We employ a purely algebraic multigrid method, known as smoothed aggregation algebraic multigrid (SAAMG) [Vanek et al. 1996], to establish the hierarchy while ensuring solvability and effectiveness at the coarser levels. In addition, we propose a filtering scheme which enables us to efficiently apply SAAMG preconditioning only to unconstrained subsystems while mitigating the overhead for preconditioning preparation as much as possible.

2 RELATED WORK

2.1 Multigrid for Fluid Simulation

Multigrid (MG) has been extensively used in various fields due to its efficiency, scalability, and parallelizability; we refer the reader to relevant textbooks [Briggs et al. 2000; Trottenberg et al. 2000] for MG basics. We focus on MG in the fluid simulation context below.

Conceptually, one of the simplest MG schemes is geometric multigrid (GMG) based on geometric coarsening and re-discretization of the fluid systems (e.g., using the 8-to-1 averaging scheme for geometry) [Chentanez and Müller 2012; Weber et al. 2015]. While GMG may work with simple boundary conditions, it can easily diverge, stagnate, and fail to converge with more complex boundary conditions and geometry that arise commonly in fluid simulation, particularly if GMG is used as a standalone solver [McAdams et al. 2010]. Using GMG as a preconditioner for CG instead can significantly improve robustness, but it is still typically necessary to perform extra smoothing over the boundaries, at a non-negligible additional cost [McAdams et al. 2010]. This vulnerability is due to the discrepancy between the coarser level geometries and the original finest level system. Unfortunately, it is generally not possible to avoid this discrepancy in practical liquid simulation scenarios. One reason is that geometric coarsening is often performed in a topology-oblivious way, merging topologically disconnected components. Ferstl et al. [2014] and Dick et al. [2016] presented geometry-aware coarsening strategies featuring cell duplication to mitigate such geometric discrepancies, although this modification compromises the simplicity and efficiency of GMG. Another key factor is that Dirichlet boundaries need to be preserved to ensure solvability at each coarser level while avoiding unsolvable rank-deficient systems due to the lack of a Dirichlet

boundary. However, preserving the Dirichlet boundaries often makes them excessively dominate the system at coarser levels, leading to deviation from the original finest level system. While GMG inherently suffers from this discrepancy issue, its purely geometric nature makes it easier to combine with spatial adaptivity [Aanjaneya et al. 2017; Ferstl et al. 2014; Setaluri et al. 2014] and domain decomposition [Liu et al. 2016], as well as to extend it to viscosity systems [Aanjaneya et al. 2019] and material point method (MPM) frameworks [Wang et al. 2020].

To avoid the discrepancy issue of GMG while simultaneously ensuring solvability, algebraic MG (AMG) forms the coarser levels based on Galerkin coarsening with prolongation and restriction operators. As such, AMG has been an attractive alternative. However, given the difficulty of efficiently building effective prolongation and restriction operators in a purely algebraic way, recent approaches that use Galerkin coarsening often still exploit the geometry of the regular grid structure to assemble the two operators, e.g., using the 8-to-1 averaging scheme for matrix entries [Cai et al. 2014; Lai et al. 2020; Shao et al. 2022; Zhang 2015]. While these geometry-inspired AMG approaches can ensure solvability, typically with systems sparser than those generated by *purely* algebraic MG schemes [Briggs et al. 2000], these schemes are unfortunately topology-oblivious and can merge algebraically-disconnected components (conceptually similar to merging of disconnected cells in GMG [Dick et al. 2016; Ferstl et al. 2014]), making the coarser levels less effective [Krishnan et al. 2013].

A purely algebraic MG formulation is free of these issues because it utilizes only algebraic connectivities to form the prolongation and restriction operators (which can be especially useful for unstructured meshes [Chentanez et al. 2007]) and never merges algebraically-disconnected components. We therefore adopt the purely algebraic approach with smoothed aggregation for further efficiency [Vanek et al. 1996]; our SAAMG preconditioning for MPRGP thus enables efficient handling of box-constrained convex QPs.

2.2 Separating Solid Boundaries

In Eulerian liquid simulation, it is often essential to enforce separating solid boundary conditions to avoid an exaggerated suction effect of liquids clinging to solid boundaries [Batty et al. 2007; Chentanez and Müller 2012; Lai et al. 2020]. For this task, Batty et al. [2007] proposed a box-constrained QP formulation (which is equivalent to the linear complementarity problem (LCP) formulation [Dostl 2009; Erleben 2013]) and solved this QP via the PATH solver [Ferris and Munson 2000], although their application was limited to small problems due to the expensive solver cost. To improve the efficiency of solving this box-constrained QP (or equivalent LCP), various approaches have since been investigated. Narain et al. [2010] and Gerszewski and Bargteil [2013] employed modified incomplete Cholesky (MIC) [Bridson 2015] preconditioning with MPRGP [Dostal and Schoberl 2005; Dostl 2009], which we call MIC-MPRGP. Chentanez and Müller [2012] presented standalone multigrid solvers accounting for non-negative pressure constraints, and Lai et al. [2020] augmented their approach using the full approximation scheme (FAS-MG) to achieve convergence rates independent of grid resolution. Andersen et al. [2017] presented a nonsmooth Newton method with line search. Inglis et al. [2017] presented a primal-dual solver, a variant of the *alternating direction method of multipliers* (ADMM). Lesser et al. [2022] proposed using projected Gauss-Seidel (PGS) with an initialization given via linear solves, as PGS itself is slow to converge.

Our method employs MPRGP, which has previously been used with MIC preconditioning by Narain et al. [2010] and Gerszewski and Bargteil [2013]; however, instead of MIC, we develop an efficient and effective SAAMG preconditioning approach specifically designed for MPRGP.

3 FLUID SOLVER

We simulate inviscid liquids based on the incompressible Euler equations on a staggered grid discretization [Bridson 2015], employing the affine particle-in-cell (APIC) framework [Jiang et al. 2015] and particle-level position correction for better volume preservation and particle distributions [Takahashi and Lin 2019]. We enforce fluid incompressibility with separating solid boundary conditions via a minimization over pressure with non-negativity constraints, similar to prior work [Batty et al. 2007; Gerszewski and Bargteil 2013; Narain et al. 2010; Takahashi and Batty 2021].

Let the discrete fluid velocity update due to pressure be

$$\mathbf{u} = \mathbf{u}^* - \Delta t \mathbf{M}_f^{-1} \mathbf{G} \mathbf{p}, \tag{1}$$

where $\mathbf{u}$ and $\mathbf{u}^*$ denote fluid velocity after and before pressure force application, respectively, Δt timestep size, $\mathbf{M}_f$ diagonal fluid mass matrix, $\mathbf{G}$ discrete gradient operator, and $\mathbf{p}$ pressure. Then an objective function, $E_f(\mathbf{p})$, for pressure projection on the fluid can be defined as

$$E_f(\mathbf{p}) = \frac{1}{2} \left\| \mathbf{u}^* - \Delta t \mathbf{M}_f^{-1} \mathbf{G} \mathbf{p} \right\|_{\mathbf{M}_f}^2. \tag{2}$$

Given the set of indices $\mathcal{S}$ for the pressure cells in contact with solids [Lai et al. 2020], we can define the separating boundary constraint function for pressure $\mathbf{s}(\mathbf{p})$ as

$$\mathbf{s}(\mathbf{p}) = \mathbf{p}_i \geq 0, \text{ if } i \in \mathcal{S}. \tag{3}$$

Alternatively, we can uniformly enforce non-negative pressures (i.e., throughout the entire liquid volume) to capture a splashy liquid effect [Gerszewski and Bargteil 2013] using

$$\mathbf{s}(\mathbf{p}) = \mathbf{p} \geq 0. \tag{4}$$

Then, our goal is to solve the following box-constrained convex QP on pressure:

$$\mathbf{p} = \underset{\mathbf{s}(\mathbf{p}) \in \mathcal{S}}{\arg\min} \hat{E}_f(\mathbf{p}), \quad \hat{E}_f(\mathbf{p}) = \frac{1}{2} \mathbf{p}^T \mathbf{A} \mathbf{p} - \mathbf{p}^T \mathbf{c}, \tag{5}$$

$$\mathbf{A} = \mathbf{G}^T \mathbf{M}_f^{-1} \mathbf{G}, \quad \mathbf{c} = \frac{1}{\Delta t} \mathbf{G}^T \mathbf{u}^*, \tag{6}$$

where $\boldsymbol{\mathcal{S}}$ denotes sets of vectors satisfying the separating solid boundary constraints, and we define $\hat{E}_f(\mathbf{p})$ for simplicity and better numerical conditioning as $\hat{E}_f(\mathbf{p}) = \left(E_f(\mathbf{p}) - \frac{1}{2} \|\mathbf{u}^*\|_{\mathbf{M}_f}^2 \right) / \Delta t^2$. If there are no non-negative pressure constraints (e.g., for fluids without free surfaces), this quadratic objective (5) can simply be optimized via a single linear solve, e.g., using SAAMG preconditioned CG (SAAMG-CG).

To solve (5), we employ an active-set method, MPRGP [Dostal and Schoberl 2005; Dostl 2009] (we refer the reader to the MPRGP implementation by Narain et al. [2010] for details.), since box-constrained convex QPs can be efficiently solved with active-set methods without performing Newton iterations [Takahashi and Batty 2021]. In addition, we accelerate MPRGP by combining it with an efficient active-set expansion technique [Kružík et al. 2020; Takahashi and Batty 2021] and using warmstarting. Furthermore, similar to CG, we use SAAMG preconditioning to accelerate the convergence of MPRGP, as detailed in Sec. 4.

4 SAAMG PRECONDITIONING FOR MPRGP

MPRGP [Dostal and Schoberl 2005; Dostl 2009] is an active-set Krylov method based on CG and can be similarly accelerated with preconditioning, e.g., using MIC. While MIC is still a popular and effective preconditioner for MPRGP [Gerszewski and Bargteil 2013; Narain et al. 2010], it has limitations in our context. MIC preconditioning is inherently serial and difficult to further

accelerate via parallelization during both preparation (incomplete factorization) and application (triangular solve). We have tested parallel left-looking incomplete Cholesky factorization and triangular solves [Naumov 2011, 2012]. However, we found it to be at least 3× slower than the efficient serial implementation by Bridson and Müller [2007] on an 8-core desktop machine, because the number of variables that can be processed in parallel is small (even compared to multicolored GS), thus failing to take full advantage of the parallel architecture [Wu et al. 2022]. In addition, MIC becomes less effective at higher resolutions and the number of required MPRGP iterations can increase significantly. While symmetric successive over-relaxation (SSOR) preconditioning [Saad 2003] can address the parallelizability limitation via red-black coloring, SSOR is essentially less effective for M-matrices compared to MIC. Thus, to address the limitations above, we present a more effective multilevel preconditioner for MPRGP.

4.1 Preconditioning Strategy

We employ a purely algebraic, topology-aware MG approach known as smoothed aggregation AMG (SAAMG) [Vanek et al. 1996] which lets us ensure solvability at coarse levels while merging only algebraically-connected components. Based on the Galerkin principle, we algebraically assemble the coarser levels with prolongation operators $\mathbf{P}$ and restriction operators $\mathbf{R}$ (typically set as $\mathbf{R} = \mathbf{P}^T$), defining the coarser level matrix $\mathbf{A}_c$ by $\mathbf{A}_c = \mathbf{P}^T\mathbf{A}_f\mathbf{P}$ given a finer level matrix $\mathbf{A}_f$. (For details, see the AMGCL implementation [Demidov 2019].) To precondition MPRGP, we use a single SAAMG V-cycle [Briggs et al. 2000; Trottenberg et al. 2000] since performing more outer iterations is typically more efficient than spending more time for preconditioning [Aanjaneya 2018].

Unlike preconditioning for CG, however, directly applying preconditioning to the entire system for MPRGP can transform the box constraints into general linear constraints [Dostl 2009], which are much more difficult to handle efficiently. One approach to precondition the QP while retaining the box constraints is to apply preconditioning only to the unconstrained subsystem and ignore the constrained variables. Naively employing this approach requires one to reassemble the unconstrained subsystem and recompute any data needed for the preconditioner (e.g., MIC preconditioning requires refactorizing the unconstrained subsystem) in each MPRGP iteration due to the changing active-sets. However, this method is too costly to perform. Narain et al. [2010] avoided reassembling and refactorizing the unconstrained subsystem in their implementation for MIC preconditioning. Specifically, their method factorizes the entire system once in each simulation step and reuses the Cholesky factors by skipping constrained variables within the triangular solves (we observed that refactoring the unconstrained subsystem is only slightly more effective, achieving around 10% fewer outer iterations). Similarly, given the high expense of preparing the SAAMG preconditioner (i.e., building $\mathbf{P}$ and $\mathbf{P}^T$ followed by assembly of $\mathbf{A}_c$ via sparse matrix-matrix multiplications at all the coarser levels), we prepare the system hierarchy only once per simulation step and reuse it, skipping the constrained variables in each MPRGP iteration. Below, we detail our preconditioning method applied to the linear preconditioning system $\mathbf{Ax} = \mathbf{b}$ (within the Krylov iterative solvers), which corresponds to the linear equation $\mathbf{Ap} = \mathbf{c}$ derived from the quadratic objective (5).

4.2 Prolongation, Restriction, and Coarsening with an Active Set

To construct the hierarchy for the entire system (without distinguishing constrained and unconstrained variables) once at each simulation step, we follow the approach of Vanek et al. [1996], repeatedly forming $\mathbf{P}, \mathbf{P}^T$, and $\mathbf{A}_c$ until $\mathbf{P}$ cannot be generated due to the lack of algebraically-connected components in the aggregation. Here, while we could use Eigen's sparse matrix-matrix multiplication [Guennebaud et al. 2010], we found that Eigen's implementation can be inefficient due to its specialized matrix format and serial operations. Instead, we use parallel matrix-matrix

multiplication storing the matrices in the compressed sparse row (CSR) format [Saad 2003], as this approach is around 5× to 15× faster in our experiments. Then, we reuse these computed matrices to process only the unconstrained subsystem in the V-cycle by filtering the components from the constrained subsystem (see Algorithm 1).

4.2.1 Active Set and Selection Matrix. Considering the linear preconditioning system $\mathbf{Ax} = \mathbf{b}$, we define an active set $\mathbf{a}$ to represent whether $\mathbf{x}_i$ is constrained or not, using the element index i such that $\mathbf{a}_i = 1$ or $\mathbf{a}_i = -1$ if $\mathbf{p}_i$ in (5) is constrained by its lower or upper bound, respectively, and $\mathbf{a}_i = 0$ otherwise. (While the formulation of our specific application has lower bounds only, we consider the general case since, e.g., frictional forces in granular media need both bounds [Andrews et al. 2022; Narain et al. 2010].) We can compute the active set $\mathbf{a}$ in each MPRGP iteration [Dostal and Schoberl 2005; Dostl 2009]. If we define a diagonal selection matrix $\mathbf{S}$ by $\mathbf{S}_{ii} = 1 - |\mathbf{a}_i|$ and grouping the unconstrained and constrained variables separately in that order, we have

$$\mathbf{S} = \begin{bmatrix} \mathbf{I}_{uu} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{bmatrix}, \tag{7}$$

where $\mathbf{I}_{uu}$ denotes the identity matrix with size equal to the number of unconstrained variables. We note that while $\mathbf{S}$ is constant during each V-cycle, $\mathbf{S}$ can change between MPRGP iterations, if different variables have become constrained or unconstrained.

4.2.2 Residual Computation. Given the residual $\mathbf{r}$ for the entire system computed by $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$, we can rearrange $\mathbf{b}$, $\mathbf{A}$, and $\mathbf{x}$ (for expositional clarity) and compute the residual for the unconstrained variables $\mathbf{r}_u$ and constrained variables $\mathbf{r}_c$ ($\mathbf{r} = [\mathbf{r}_u^T, \mathbf{r}_c^T]^T$) as

$$\mathbf{r} = \mathbf{Sb} - \mathbf{S}^T\mathbf{ASx}, \tag{8}$$

where

$$\mathbf{b} = \begin{bmatrix} \mathbf{b}_u \\ \mathbf{b}_c \end{bmatrix}, \mathbf{A} = \begin{bmatrix} \mathbf{A}_{uu} & \mathbf{A}_{uc} \\ \mathbf{A}_{cu} & \mathbf{A}_{cc} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} \mathbf{x}_u \\ \mathbf{x}_c \end{bmatrix}, \tag{9}$$

and $\mathbf{r}_c = 0$. This procedure essentially works as a filtering mechanism and is also used in (11), (12), and (13) to eliminate any constraint violation by construction; related techniques have been proposed in prior work to preprocess collision handling in cloth [Ascher and Boxerman 2003; Baraff and Witkin 1998; Tamstorf et al. 2015].

4.2.3 Active-Set Restriction. Considering the changing active set at the finest level $\mathbf{a}_f$ in each MPRGP iteration, we recompute the coarser level active set $\mathbf{a}_c$ based on $\mathbf{a}_f$. As the coarser level solution $\mathbf{x}_c$ must be treated as constrained to ensure convergence if an element in $\mathbf{x}_c$ is related to any constrained elements in the finer solution $\mathbf{x}_f$ via the restriction operator $\mathbf{P}^T$, we define the coarser level active-set $\mathbf{a}_c$ by

$$\mathbf{a}_{c,i} = \begin{cases} 1 & \text{if } 0 < (\mathbf{P}^T)_i\mathbf{a}_f, \\ -1 & \text{if } 0 > (\mathbf{P}^T)_i\mathbf{a}_f, \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

where $\mathbf{a}_{c,i}$ denotes the ith element of $\mathbf{a}_c$, and $(\mathbf{P}^T)_i$ the ith row of $\mathbf{P}^T$. While we can simplify the computation (10) to determine $\mathbf{a}_c$ since our formulation has lower bounds only, this definition enables us to naturally handle both lower and upper bounds. Once we set the coarser level active set $\mathbf{a}_c$, we can compute the coarser level selection matrix $\mathbf{S}_c$, as done for $\mathbf{S}_f$.

4.2.4 Residual Restriction. The residual computed at a finer level $\mathbf{r}_f$ can be restricted to the right hand side vector at the coarser level (i.e., $\mathbf{b}_c$), by multiplying the restriction operator $\mathbf{P}^T$ while filtering the constrained variable at both levels using

$$\mathbf{b}_c = \mathbf{S}_c\mathbf{P}^T\mathbf{S}_f\mathbf{r}_f. \tag{11}$$

4.2.5 Error Correction via Prolongation. Similarly, the coarser level solution $\mathbf{x}_c$ can be interpolated via the prolongation operator $\mathbf{P}$ to correct the solution at the finer level $\mathbf{x}_f$ while filtering the constrained variables at both levels using

$$\mathbf{x}_f = \mathbf{x}_f + \mathbf{S}_f\mathbf{P}\mathbf{S}_c\mathbf{x}_c. \tag{12}$$

4.3 Smoother

For smoothing, we adopt an approach based on sparse approximation inverses (SPAI) [Grote and Huckle 1997]. In particular, we employ SPAI-0 as a smoother [Bröker et al. 2001]; since this variant uses a diagonal matrix, it is embarrassingly parallel (like damped Jacobi), and Bröker et al. [2001] showed it to be more effective than damped Jacobi even with an optimal damping parameter. This parallelizability is essential because SAAMG sacrifices the regular grid structure (which enables the simple red-black coloring used in red-black GS (RBGS)) and would require expensive coloring at the coarser levels for GS-type smoothers, although in practice we did not observe a noticeable convergence advantage of multicolored GS over SPAI-0 for SAAMG.

The update of the SPAI-0 smoother (and other stationary iterative methods) can be written in the form $\mathbf{x}^{k+1} = \mathbf{x}^k + \omega\mathbf{W}(\mathbf{b} - \mathbf{A}\mathbf{x}^k)$, where k denotes an iteration index, ω a relaxation factor ($\omega = 1$ for SPAI-0), and $\mathbf{W}$ a sparse inverse approximation of $\mathbf{A}$ [Bröker et al. 2001]. Thus, we can perform the smoothing for the unconstrained variables at each level as

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \omega\mathbf{W}(\mathbf{S}\mathbf{b} - \mathbf{S}^T\mathbf{A}\mathbf{S}\mathbf{x}^k). \tag{13}$$

As the smoothing operation includes the form of the residual, we can also write (13) as $\mathbf{x}^{k+1} = \mathbf{x}^k + \omega\mathbf{W}\mathbf{r}^k$ using the residual computation (8). Note that constraints on $\mathbf{x}$ are automatically addressed by the filtering mechanism and actual values of $\mathbf{x}$ are not bounded by the box constraints on $\mathbf{p}$. At the coarsest level, we perform SPAI-0 smoothing similar to the other levels.

As we use SAAMG as a preconditioner for MPRGP, we need to set our initial guess as 0 at each level in the V-cycle [McAdams et al. 2010; Tatebe 1993] and utilize this fact and parallelizability of SPAI-0 to save the smoothing cost by removing the first matrix-vector multiplication in the pre-smoothing [Adams et al. 2003; McAdams et al. 2010].

For smoothing, we perform only one pre- and post-smoothing at each level because more smoothing did not improve the total computational time for convergence if the V-cycle is used as a preconditioner, although the total number of outer iterations decreases [Dick et al. 2016]. While there are other embarrassingly parallel smoothers, e.g., scheduled relaxed Jacobi (SRJ) [Yang and Mittal 2017] and polynomial Chebyshev [Adams et al. 2003], these approaches are defined only with more than one smoothing iteration (as a single smoothing iteration corresponds to damped Jacobi), and hence spend more time for smoothing. In practice, performing more smoothing iterations sacrifices the opportunity to utilize the zero-initialization, and we observed that SPAI-0 is more efficient than these smoothers. Table 1 shows performance evaluations for various smoothers.

4.4 Our Preconditioning Algorithm

Algorithm 1 shows our procedure for SAAMG V-cycle preconditioning for MPRGP. As the active set $\mathbf{a}$ and thus $\mathbf{S}$ change in each MPRGP iteration (i.e., in each V-cycle, since we use only one V-cycle for preconditioning), we handle $\mathbf{S}$-related computations in (8), (11), (12), and (13) on the fly

without explicitly forming S-related terms and rearranging $\mathbf{b}$, $\mathbf{A}$, and $\mathbf{x}$, given the expensive cost of rearranging $\mathbf{A}$ in the CSR format. We also provide illustrative operation examples for (8) and (10) in Appendix B.

Algorithm 1 V-cycle($\mathbf{A}_f$, $\mathbf{x}_f$, $\mathbf{b}_f$, $\mathbf{a}_f$)

```
Initialize x_f = 0
if Coarsest
    Pre- and post-smoothing with (13)
else
    Pre-smoothing with (13)
    Compute a_c with (10) and b_c with (8) and (11)
    V-cycle(A_c, x_c, b_c, a_c)
    Update x_f with (12)
    Post-smoothing with (13)
end if
```

5 RESULTS AND DISCUSSIONS

We implemented and parallelized our method in C++ with OpenMP. All examples used 60 frames per second and adaptive timestepping with CFL numbers between 3.0 and 5.0 (empirically chosen per scene based on energy preservation and visual quality). In the preconditioning evaluations using fluids without free surfaces, i.e., pure Neumann boundary problems (Figures 2 and 3), we ensure the compatibility condition (which is automatically satisfied unless the simulation settings are incompatible) [Bridson 2015] and eliminate the null space by explicitly removing (pinning) one pressure degree of freedom to guarantee solver convergence. In our experiments for fluids with free surfaces, we use (3) for separating solid boundary conditions, unless otherwise stated. We use an absolute or relative residual tolerance of 10^{-8} as our termination criteria, unless otherwise stated. To visualize fluids without free surfaces, we render traces of passively advected particles over 5-10 frames. We executed all the simulations on a desktop machine with an Intel Core i7-9700 (8 cores) with 16GB RAM.

5.1 Evaluation of Smoothers

We evaluate various smoothing schemes to justify our use of the SPAI-0 smoother [Bröker et al. 2001] within SAAMG-CG, using a 3D cubic domain, where a source term is placed on the domain center while the outermost boundaries are fixed with a zero pressure Dirichlet boundary condition. We compare the following schemes:

(1) Damped Jacobi (1 iter): [Trottenberg et al. 2000];
(2) Damped Jacobi (2 iters);
(3) SPAI-0 (1 iter): [Bröker et al. 2001];
(4) SPAI-0 (2 iters);
(5) Chebyshev (2 iters): [Adams et al. 2003];
(6) SRJ (2 iters): scheduled relaxed Jacobi [Yang and Mittal 2017];
(7) RBGS + SPAI-0 (1 iter).

We use the same number of pre- and post-smoothing steps, as noted above in the parentheses beside each scheme. For damped Jacobi, we use the optimal parameter $\omega = 6/7$ [Briggs et al. 2000; Trottenberg et al. 2000]. For Chebyshev, we evaluate the spectral radius based on the Gershgorin circle theorem. For RBGS + SPAI-0, we use RBGS for the finest level utilizing the regular grid

Table 1. Evaluation of smoothers within SAAMG-CG. Numbers represent the hierarchy construction time + CG solve time (CG iteration count) in seconds. SPAI-0 (1 iter) is most efficient, as emphasized in bold.

Smoother \ Resolution	64^3	128^3	256^3
Damped Jacobi (1 iter)	0.06+0.12 (17)	0.56+1.16 (19)	3.89+11.77 (23)
Damped Jacobi (2 iters)	0.06+0.13 (13)	0.51+1.40 (15)	3.76+13.32 (17)
SPAI-0 (1 iter)	0.06+0.11 (16)	0.52+1.10 (18)	3.76+**11.12** (22)
SPAI-0 (2 iters)	0.06+0.11 (12)	0.51+1.33 (14)	3.87+13.31 (17)
Chebyshev (2 iters)	0.06+0.19 (17)	0.51+2.01 (19)	3.83+17.64 (19)
SRJ (2 iters)	0.07+0.13 (12)	0.52+1.31 (14)	3.73+12.50 (16)
RBGS + SPAI-0 (1 iter)	0.06+0.14 (16)	0.51+1.46 (18)	3.80+14.29 (21)

structure and SPAI-0 for the coarser levels. We use a termination relative residual of 10^{-10}. Table 1 summarizes the evaluation settings and results.

In our experiments, while most smoothers performed comparably, SPAI-0 (1 iter) was most efficient, and this result agrees with the claim that parameter-free SPAI-0 is more effective than damped Jacobi with optimal damping parameter [Bröker et al. 2001]. The Chebyshev smoother was less effective due to the relatively expensive operations required, and the iteration count was not sufficiently reduced due to the inaccurate spectral radius estimation. Although SRJ can reduce the number of iterations more than SPAI-0 can, SRJ was not fastest due to the need for at least 2 smoothing iterations. It is typically more efficient to perform more CG iterations with minimal time spent on preconditioning [Aanjaneya 2018]. In addition, more smoothing iterations spoil the opportunities to take advantage of zero-initialization (e.g., we can save 50% of the smoothing cost with one smoothing iteration vs. only 25% with two smoothing iterations). RBGS + SPAI-0 was also not faster than SPAI-0 because using RBGS only at the finest level did not particularly improve the convergence and also reducing opportunities to take advantage of zero-initialization [Adams et al. 2003].

5.2 Evaluation of Aggregation Schemes

To justify our choice of purely algebraic smoothed aggregation [Vanek et al. 1996], we evaluate multiple aggregation schemes for MG-preconditioned CG. We consider fluids without free surfaces in a maze-like domain (with purely Neumann boundaries) with continuously added external forces from the bottom, discretized with a grid resolution of 128^3, as shown in Figure 2. We compare the following aggregation schemes, along with the baseline MICCG (which is more efficient than ICCG for Poisson equations [Bridson 2015; Dick et al. 2016]):

(1) MICCG: baseline [Bridson 2015];
(2) GIUA: geometry-inspired unsmoothed aggregation [Shao et al. 2022; Zhang 2015];
(3) PAUA: purely algebraic unsmoothed aggregation (e.g., [Chentanez et al. 2007]);
(4) PASA (ours): purely algebraic smoothed aggregation [Vanek et al. 1996].

For GIUA, we use the 8-to-1 averaging scheme [Shao et al. 2022; Zhang 2015] and employ RBGS smoothing taking advantage of the regular grid structure (as we also observed that RBGS was more efficient than damped Jacobi and SPAI-0 smoothers with GIUA [Shao et al. 2022]). To accelerate the convergence of the unsmoothed aggregation schemes (GIUA and PAUA), we employ overinterpolation [Stüben 2001]. Figure 2 compares profiles of the computational cost for the entire pressure solve phase (including valid cell evaluations, operator construction, system assembly, and system solve to show overall performance benefit), and Table 2 summarizes the simulation settings and averaged results.

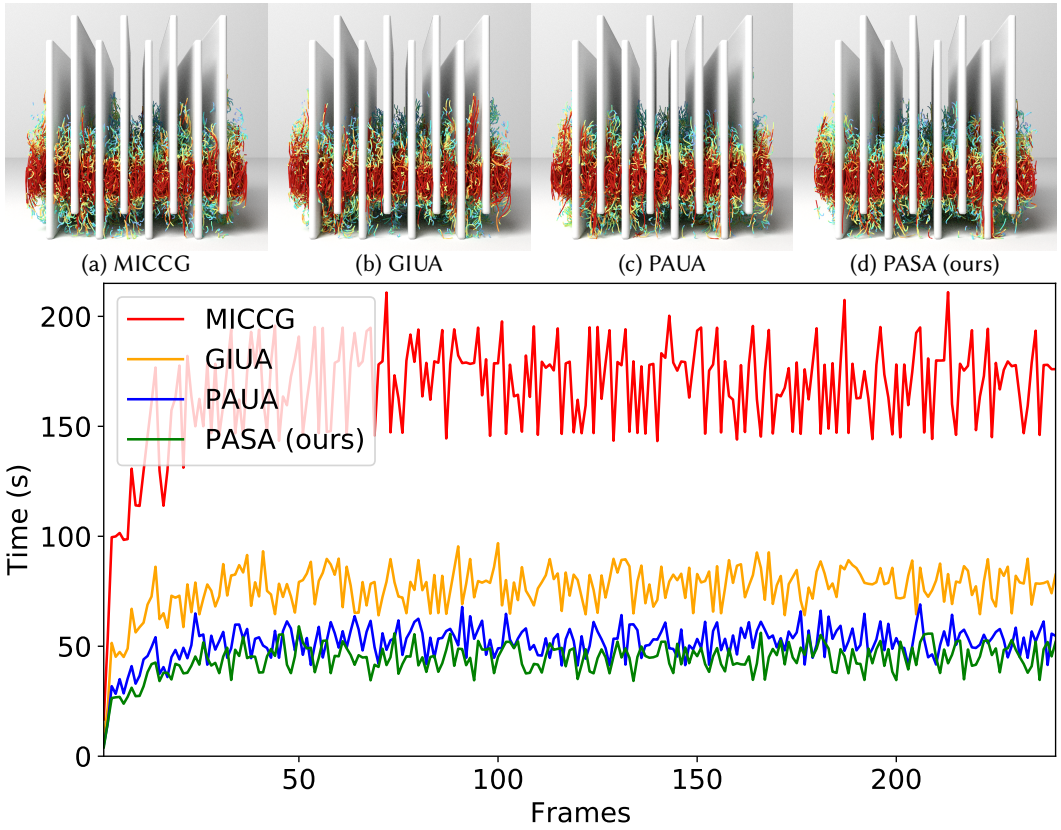

Fig. 2. (Top) Fluids in a closed maze-like domain with continuously added external forces, simulated with different approaches (see Sec. 5.2). Tracer particles are color-coded based on the fluid velocities. (Bottom) Profiles of the total time for the entire pressure solve phase per frame. While all the approaches generate comparable visual results, PASA (ours) achieves 3.8×, 1.7×, and 1.2× faster performance than MICCG, GIUA, and PAUA, respectively.

Table 2. Simulation settings and results for Figure 2. The system size is denoted by n, the number of CG iterations per solve by N, the number of substeps s, and total time (s) per frame by T.

Scheme	n	N	s	T
MICCG	1.3M	488.2	10.3	166.8
GIUA	1.3M	99.2	10.3	76.3
PAUA	1.3M	52.5	10.3	51.5
PASA (ours)	1.3M	16.7	10.4	44.2

Despite the difficulty of this pure Neumann problem, these schemes were able to fully converge and generated comparable visual results (with only minor differences due to the accumulated effects of slightly distinct residuals at solver termination). However, as GIUA merges algebraically-disconnected components, the effectiveness of the preconditioning is significantly degraded, thus requiring many more CG iterations to converge. By contrast, due to its purely algebraic formulation, PAUA only merges algebraically-connected components, leading to more effective preconditioning.

In addition, smoothing the prolongation/restriction operators improves the effectiveness of the preconditioner, further reducing the number of necessary CG iterations (see Table 2). Although the reduction in CG iterations is not *proportionally* reflected in the overall solver performance, due to the system and hierarchy construction overhead and more expensive per V-cycle cost, PASA nevertheless achieves 3.8×, 1.7×, and 1.2× faster performance than MICCG, GIUA, and PAUA, respectively.

To roughly evaluate the performance at higher resolutions, we experimented with GIUA and PASA using a grid resolution of 192^3 over 5 frames. GIUA took 190.4s per frame on average, while PASA took 92.7s, thus exhibiting around 2.1× faster performance. This result clearly suggests that PASA scales better than GIUA, and thus PASA is likely to be increasingly beneficial at even higher resolutions. Furthermore, we believe that the purely algebraic approaches, PAUA and PASA, will also be advantageous over GIUA in more complex scenarios, where algebraically-disconnected components can merge, e.g., for fluid-related topology optimization [Li et al. 2022; Liu et al. 2022].

5.3 Preconditioning vs. Standalone MG

To demonstrate that using SAAMG as a preconditioner for Krylov iterative solvers is at least as efficient as standalone MG solvers, we compare SAAMG-CG with standalone SAAMG linear solvers, using fluids without free surfaces in a closed domain (with purely Neumann boundaries) with continuously added external forces, discretized with a grid resolution of 128^3, as shown in Figure 3. We evaluate the following schemes:

(1) SAAMG-V: standalone V-cycle linear solver (Algorithm 1 with no zero-initialization (line 1) and no active sets);
(2) SAAMG-FAS: V-cycle FAS specialized to solving a linear system (Algorithm 2 in Appendix A);
(3) SAAMG-CG (ours).

For SAAMG-V, we perform 5 pre- and post-smoothing steps as this choice achieved the best cost performance (using fewer than 4 iterations diverged while using 4 iterations required many more V-cycles for convergence). Figure 3 compares profiles of computational costs for the entire pressure solve phase, and Table 3 summarizes the simulation settings and averaged results.

Table 3. Simulation settings and results for Figure 3. The system size is denoted by n, the number of CG/V-cycle iterations per solve by N, the number of substeps by s, and total time (s) per frame by T.

Scheme	n	N	s	T
SAAMG-V	1.6M	30.3	8.3	64.8
SAAMG-FAS	1.6M	20.4	8.2	42.0
SAAMG-CG (ours)	1.6M	11.6	8.3	38.8

In this comparison, these schemes were able to fully converge and all generated comparable visual results. However, SAAMG-V with 5 smoothing iterations still required a larger number of outer iterations compared to SAAMG-FAS and SAAMG-CG. SAAMG-FAS improves the robustness of the V-cycle iterations by focusing on the difference from the current approximation. However, FAS is not designed to be used as a preconditioner for Krylov iterative solvers (see Appendix A), and thus SAAMG-FAS cannot be combined with MPRGP to handle box constraints. SAAMG-CG stably and quickly converged due to the preconditioning provided by the V-cycle, and achieves 1.7× and 1.1× faster performance than SAAMG-V and SAAMG-FAS, respectively, demonstrating that a SAAMG-preconditioned Krylov solver is at least as efficient as standalone SAAMG linear solvers.

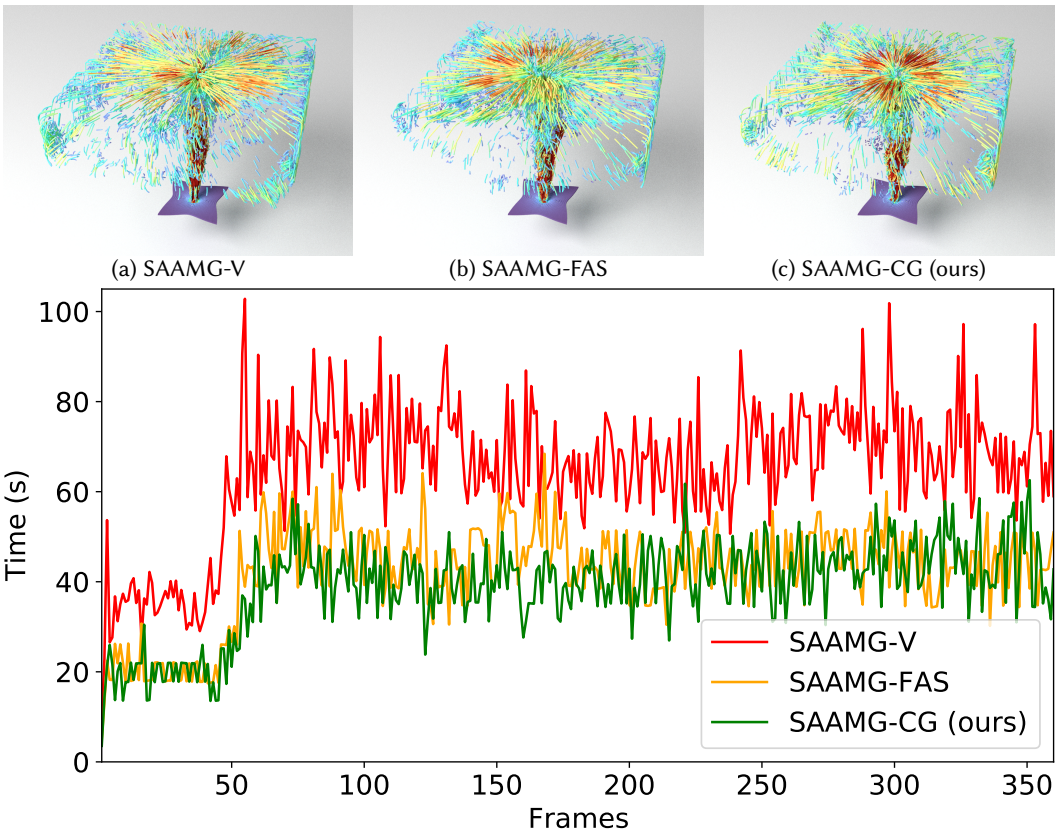

Fig. 3. (Top) Fluids in a closed domain with continuously added external forces, simulated with different pressure solvers (see Sec. 5.3). Tracer particles are color-coded based on the fluid velocities. (Bottom) Profiles of the total time for the entire pressure solve phase per frame. While all of these schemes generate comparable visual results, SAAMG-CG (ours) achieves 1.7× and 1.1× faster performance than SAAMG-V and SAAMG-FAS, respectively.

5.4 Separating Solid Boundaries

To demonstrate the effectiveness of our SAAMG-MPRGP scheme, we compare it with various box-constrained convex QP solvers for separating solid boundary conditions in two scenarios: sphere and maze.

5.4.1 Sphere. We experiment with a liquid volume in a solid sphere using a grid resolution of 128^3 with 1.7M particles, as shown in Figure 4. We compare the following schemes:

(1) SAAMG-CG: baseline;
(2) SAAMG-CG+PRBGS: projected RBGS (PRBGS) with an initial guess given by SAAMG-CG (e.g., [Lesser et al. 2022]);
(3) PI+SAAMG-CG: policy iteration [Lai et al. 2020] with inner SAAMG-CG solves;
(4) SAAMG-MPRGP (ours).

Although the work of Lai et al. [2020] recommended FAS for handling separating solid boundary conditions, in our experiments we found that FAS diverged and so we instead used their proposed policy iteration method [Lai et al. 2020], combining it with SAAMG-CG. Following Lai's approach,

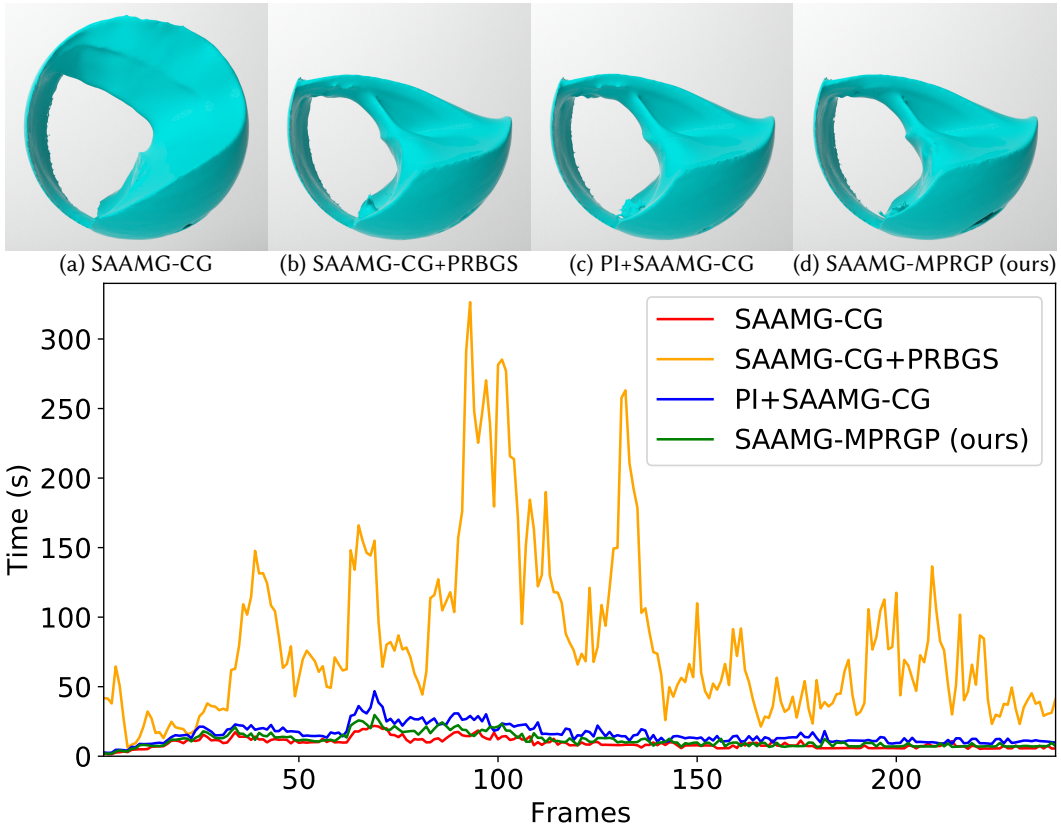

Fig. 4. (Top) Liquid in a solid sphere (not rendered), simulated with various solvers (see Sec. 5.4). (Bottom) Profiles of the total time for the entire pressure solve phase per frame. While all of these solvers generate comparable results (except for the baseline SAAMG-CG), SAAMG-MPRGP (ours) is 7.1× and 1.4× faster than SAAMG-CG+PRBGS and PI+SAAMG-CG, respectively.

Table 4. Simulation settings and results for Figure 4. The system size is denoted by n, the number of PRBGS or PI iterations by M, the number of CG/MPRGP iterations per solve by N, the number of substeps by s, and the total time (s) per frame by T.

Scheme	n	M	N	s	T
SAAMG-CG	337.2k		12.8	4.8	9.4
SAAMG-CG+PRBGS	333.4k	2.7k	12.8	4.9	82.0
PI+SAAMG-CG	331.4k	6.0	11.3	4.8	15.9
SAAMG-MPRGP (ours)	329.7k		42.2	4.9	11.6

we explicitly removed the constrained variables to avoid solver stagnation due to numerical issues, forming smaller unconstrained systems within each policy iteration. Figure 4 compares profiles of the computational cost for the entire pressure solve phase, and Table 4 summarizes the simulation settings and averaged results.

All of SAAMG-CG+PRBGS, PI+SAAMG-CG, and SAAMG-MPRGP generated plausible liquid behaviors, unlike SAAMG-CG. SAAMG-CG+PRBGS required many PRBGS iterations, especially

Table 5. Simulation settings and results for Figure 5. The system size is denoted by n, the number of PI iterations by M, the number of CG/MPRGP iterations per solve by N, the number of substeps per frame by s, and total time (s) per frame by T.

Scheme	n	M	N	s	T
SAAMG-CG	1.2M		13.3	12.6	103.3
MIC-MPRGP	1.2M		474.6	12.1	303.2
PI+SAAMG-CG	1.2M	11.2	13.9	12.2	208.1
SAAMG-MPRGP (ours)	1.2M		59.0	12.1	123.0

when the initial guess given by SAAMG-CG was not accurate (i.e., when fluids are separating from the solid), and this approach was much more costly compared to the other schemes. The policy iteration in PI+SAAMG-CG can be considered as active-set updates, and thus this approach is close to our SAAMG-MPRGP. One key difference lies in how frequently active sets are updated. As PI+SAAMG-CG updates active sets after each SAAMG-CG solve, the active sets are less frequnetly updated, wasting the computational efforts for the inner solve. By contrast, our SAAMG-MPRGP updates the active sets in each MPRGP iteration and thus achieves faster convergence due to the up-to-date active sets. Consequently, our SAAMG-MPRGP is 1.4× faster than PI+SAAMG-CG.

5.4.2 Maze. Next, we evaluate our SAAMG-MPRGP in the maze scenario using the grid resolution of 128^3 with 8.2M particles, as shown in Figure 5. This scenario faces larger condition numbers and thus is more difficult to solve, compared to the scenario in Figure 4, due to the many Neumann (solid) boundaries dominating over the Dirichlet (free surface) boundaries. We compare the following schemes:

(1) SAAMG-CG: baseline;
(2) MIC-MPRGP: [Narain et al. 2010];
(3) PI+SAAMG-CG: [Lai et al. 2020];
(4) SAAMG-MPRGP (ours).

Figure 5 compares profiles of the computational costs for the entire pressure solve phase, and Table 5 summarizes the simulation settings and averaged results.

While all of MIC-MPRGP, PI+SAAMG-CG, and SAAMG-MPRGP generated comparable results correctly supporting the wall separation of liquids, our SAAMG-MPRGP outperformed MIC-MPRGP and PI+SAAMG-CG by factors of 2.5 and 1.7, respectively. Considering the scaling of MIC preconditioning and policy iterations (as the required number of PI iterations is higher here compared to Figure 4), we believe that our SAAMG-MPRGP will be more advantageous for ill-conditioned, higher-resolution problems.

5.5 Splashy Liquids

Lastly, we evaluate the efficiency of our method on the splashy liquids formulation [Gerszewski and Bargteil 2013] given by (4) in the dambreak scenario, discretized with a grid resolution of 96^3 with 1.7M particles, as shown in Figure 6. We compare the following schemes:

(1) SAAMG-CG: baseline;
(2) SAAMG-MPRGP: baseline with (3) ("non-splashy");
(3) MIC-MPRGP: [Narain et al. 2010];
(4) PI+SAAMG-CG: [Lai et al. 2020];
(5) SAAMG-MPRGP (ours).

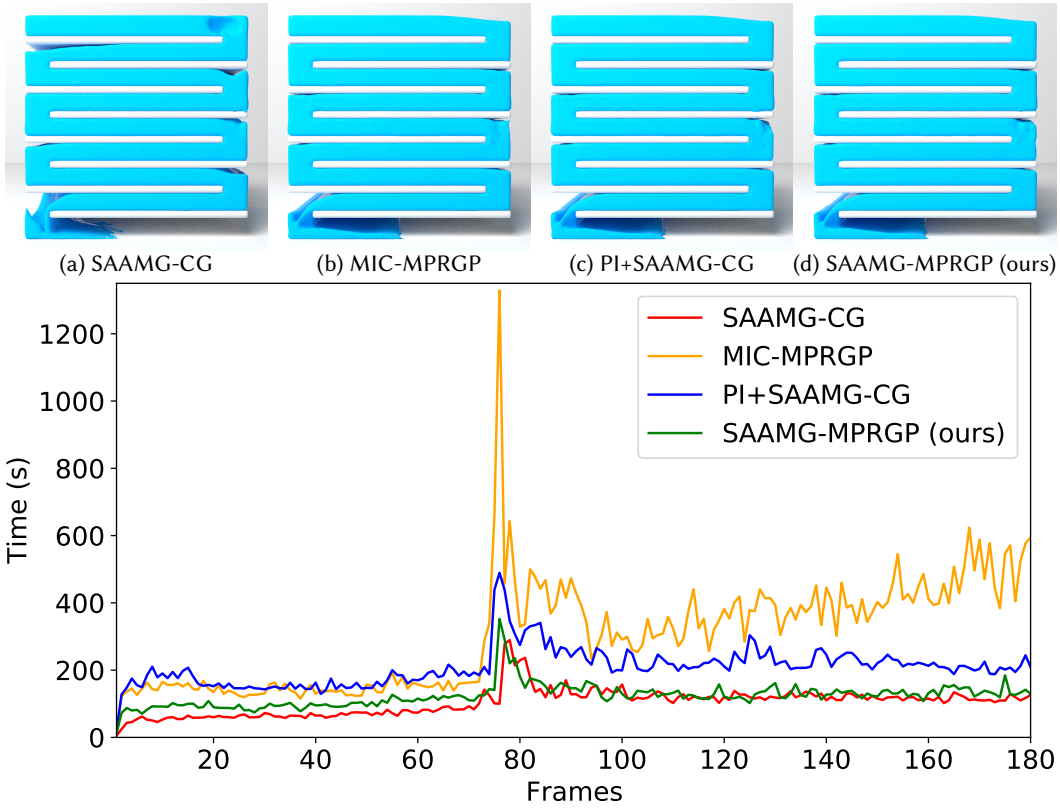

Fig. 5. (Top) Liquid in a maze, simulated with various solvers (see Sec. 5.4). SAAMG-CG exhibits artifacts of liquid suction to the solid wall at the top right and bottom left parts while the others do not. (Bottom) Profiles of the total time for the entire pressure solve phase per frame. While all of these solvers generate comparable results (except SAAMG-CG), SAAMG-MPRGP (ours) is 2.5× and 1.7× faster than MIC-MPRGP and PI+SAAMG-CG, respectively.

Table 6. Simulation settings and results for Figure 6. The system size is denoted by n, the number of PI iterations by M, the number of CG/MPRGP iterations per solve by N, the number of substeps per frame by s, and total time (s) for the system solve and entire pressure solve phase per frame by T_s and T, respectively.

Scheme	n	M	N	s	T_s	T
SAAMG-CG	318.4k		12.2	4.4	0.4	6.9
SAAMG-MPRGP with (3)	337.7k		27.1	4.7	1.5	8.0
MIC-MPRGP	350.3k		123.4	4.7	5.3	11.5
PI+SAAMG-CG	348.9k	9.0	11.4	4.8	8.5	15.1
SAAMG-MPRGP (ours)	349.4k		35.9	4.6	1.9	8.4

Figure 6 compares profiles of computational costs for the entire pressure solve phase (middle) and, separately, the system solve only (i.e., excluding valid cell evaluations, operator construction, and system assembly) to show the pure performance gain of our method (bottom). Table 5 summarizes the simulation settings and averaged results.

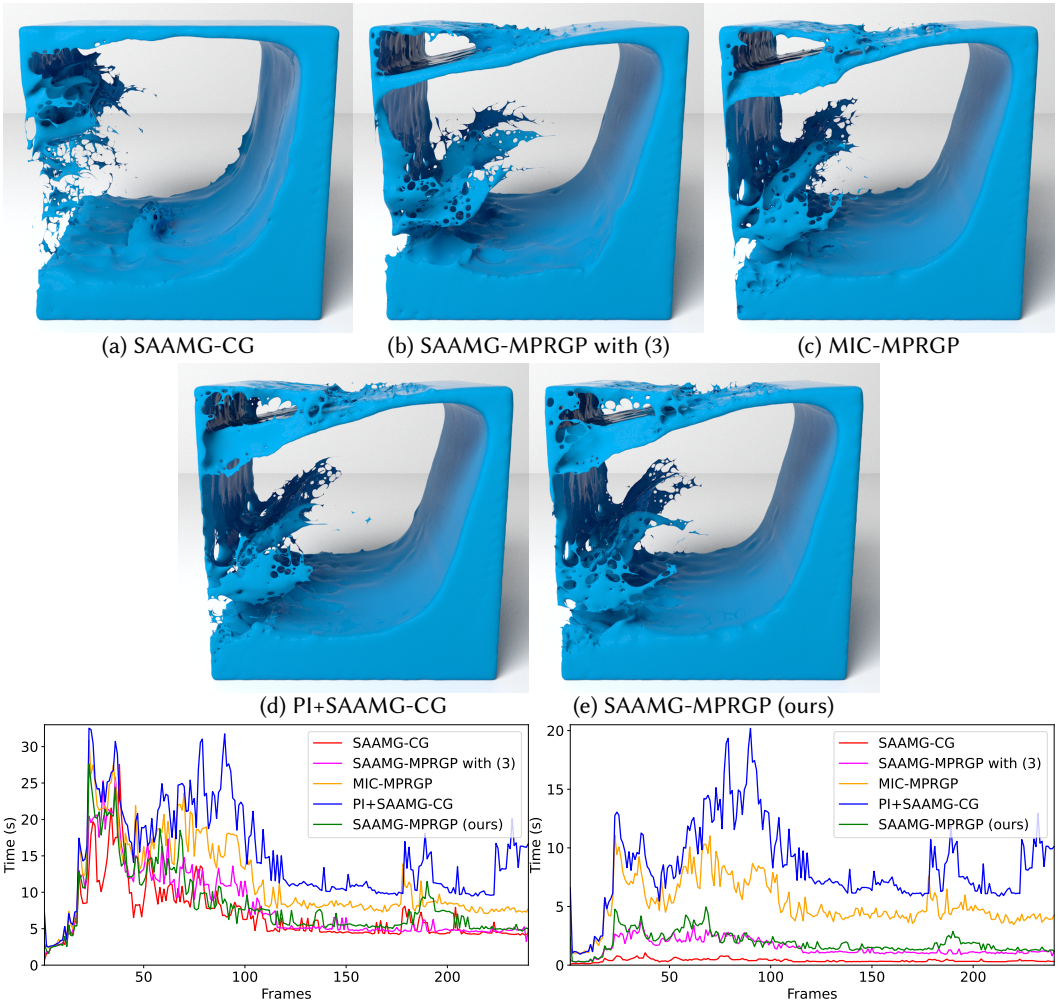

Fig. 6. (Top) Splashy liquid dam break, simulated with various solvers (see Sec. 5.5). (Middle and Bottom) Profiles of the total time per frame for the entire pressure solve (middle) and system solve only (bottom). Except for SAAMG-CG and SAAMG-MPRGP with (3), all of these solvers generate comparable visual results. SAAMG-MPRGP (ours) is 1.4× and 1.8× faster than MIC-MPRGP and PI+SAAMG-CG for the entire pressure solve phase, respectively, and is 2.8× and 4.5× faster than MIC-MPRGP and PI+SAAMG-CG for the system solve itself, respectively.

Except for the SAAMG-CG and SAAMG-MPRGP with (3), all the other solvers (MIC-MPRGP, PI+SAAMG-CG, and our SAAMG-MPRGP) generated plausible and comparable visual results for splashy liquids with separating solid boundaries. In this dambreak example, there are more Dirichlet boundaries compared to the scene in Fig. 5 (so it is easier to solve), and MIC preconditioning was sufficiently effective. As a result, MIC-MPRGP was faster than PI+SAAMG-CG. In this comparison, our SAAMG-MPRGP achieved 1.4× and 1.8× faster performance than MIC-MPRGP and PI+SAAMG-CG, respectively, for the entire pressure solve phase while ours is 2.8× and 4.5× faster than MIC-MPRGP and PI+SAAMG-CG, respectively, for the pure system solve part, due to the more effective

preconditioning and frequent active-set updates enabled with our efficient filtering scheme. In addition, the comparison of SAAMG-MPRGP with (3) vs. (4) (i.e., inequality conditions only at boundaries vs. everywhere) shows that our SAAMG-MPRGP can effectively handle additional box constraints with a relatively small extra cost.

5.6 Discussion

5.6.1 Box-Constrained Convex QP vs. LCP. While the non-negative pressure constraints for the separating solid boundary conditions and splashy liquids can also be encoded as an LCP [Erleben 2013] (or nonlinear equations [Lai et al. 2020]), we prefer formulating these as a box-constrained convex QP because the minimization formulation enables us to use various numerical optimization techniques and solvers [Dostl 2009; Nocedal and Wright 2006] and to easily combine other non-quadratic objective terms, e.g., for two-way coupling with elastic solids [Takahashi and Batty 2022] and position-level frictional contacts [Li et al. 2020].

5.6.2 Geometry-Inspired AMG. In our framework, we employed a purely algebraic approach for the hierarchy construction because of its robustness and generality, as demonstrated in Figure 2. However, geometry-inspired AMG approaches can also ensure the solvability at the coarser levels due to the Galerkin principle with sparser systems which contribute to better performance and smaller memory usage in some cases. In addition, it would be possible to utilize the preserved regular grid structures for the red-black coloring enabling over-relaxation [Shao et al. 2022; Zhang 2015], pressure extrapolation [Lai et al. 2020], boundary-aware triangular interpolation [Lai et al. 2020], and matrix-free implementation reducing the hierarchy construction overhead and memory footprint [Shao et al. 2022]. As such, it is promising to explore an algebraic approach that utilizes regular grid structures while retaining them in the course of the hierarchy construction.

5.6.3 SAAMG Preconditioning. Simple stationary iterative methods (e.g., damped Jacobi, GS, and SPAI-0) are typically sufficient as a smoother for our pressure Poisson system involving the Laplacian matrix (which is an M-matrix) to achieve nearly linear scaling with respect to number of unknowns. However, for more challenging systems (e.g., viscosity [Aanjaneya et al. 2019; Shao et al. 2022] and elasticity [Chen et al. 2021; Zhu et al. 2010]), stronger smoothers (e.g., box smoother, SPAI-1, or incomplete LU) are often necessary to achieve linear scaling, although these smoothers are typically much more expensive (e.g., SPAI-1 was around 10× and 2× more costly to prepare and apply, respectively, compared to SPAI-0 using AMGCL [Demidov 2019]).

In addition, our system with its M-matrix is isotropic, aggregation-based approaches using the system matrix coefficients are sufficient to build topology-aware prolongation operators. However, for anisotropic systems with non-M-matrices (e.g., due to contacts and variable density, viscosity, and elasticity parameters), it would be essential to take into account the algebraic smoothness of the system [Briggs et al. 2000; Trottenberg et al. 2000]. As such, it appears promising to investigate more effective AMG methods, e.g., extending the coarsening scheme specialized for the Laplacian matrix proposed by Krishnan et al. [2013].

5.6.4 SAAMG-MPRGP. Our box-constrained convex QP solver, SAAMG-MPRGP, is general and can be applied to other problems, e.g., frictional contact handling [Andrews et al. 2022], granular flows [Narain et al. 2010; Takahashi and Batty 2021], skinning using non-negative least squares [James and Twigg 2005] and bounded biharmonic weighting [Jacobson et al. 2011], cubature optimization [An et al. 2008], and level-set smoothing [Bhattacharya et al. 2015]. Thus, it is worthwhile evaluating how SAAMG-MPRGP performs in different settings.

Due to the frequent updates of the active sets in MPRGP and our preconditioning strategy applied only to the unconstrained variables, we cannot take full advantage of SAAMG preconditioning as

the number of MPRGP iterations required for convergence is larger than those for CG. Thus, it would be interesting to investigate interior point methods that do not use active sets [Nocedal and Wright 2006] and explore conjugate projectors to precondition constrained variables [Dostl 2009].

5.6.5 Surface-Tension. As our focus is on large scale liquid behaviors, we did not consider surface tension forces. In addition, non-negative pressure constraints eliminate surface-tension-like attractive and adhesive forces. As such, one would need to explicitly model surface forces to achieve suction effects.

6 CONCLUSIONS

We have proposed a new multilevel active-set preconditioning scheme based on SAAMG and combined it with MPRGP to develop a powerful box-constrained convex QP solver, SAAMG-MPRGP which can efficiently enforce fluid incompressiblity while handling non-negative pressure constraints for separating solid boundary conditions in liquid animation. Our approach employs a purely algebraic coarsening strategy to guarantee the solvability of the coarser level systems via the Galerkin principle and ensures the effectiveness of the hierarchy by merging only algebraically-connected components. By reusing the coarser level systems assembled once per simulation step, we efficiently apply the preconditioning to the unconstrained subsystems for MPRGP using our filtering scheme. We demonstrated the efficacy of our SAAMG-MPRGP over prior methods in various examples.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable suggestions and comments. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (Grant RGPIN-2021-02524).

REFERENCES

Mridul Aanjaneya. 2018. An Efficient Solver for Two-way Coupling Rigid Bodies with Incompressible Flow. *Computer Graphics Forum* 37, 8 (2018), 59–68.

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 (jul 2017), 12 pages.

Mridul Aanjaneya, Chengguizi Han, Ryan Goldade, and Christopher Batty. 2019. An Efficient Geometric Multigrid Solver for Viscous Liquids. *Proc. ACM Comput. Graph. Interact. Tech.* 2, 2, Article 14 (July 2019), 21 pages.

Mark Adams, Marian Brezina, Jonathan Hu, and Ray Tuminaro. 2003. Parallel Multigrid Smoothing: Polynomial versus Gauss–Seidel. *J. Comput. Phys.* 188, 2 (jul 2003), 593–610.

Steven S. An, Theodore Kim, and Doug L. James. 2008. Optimizing Cubature for Efficient Integration of Subspace Deformations. *ACM Trans. Graph.* 27, 5, Article 165 (dec 2008), 10 pages.

Michael Andersen, Sarah Niebe, and Kenny Erleben. 2017. A Fast Linear Complementarity Problem (LCP) Solver for Separating Fluid-Solid Wall Boundary Conditions. In *Proceedings of the 13th Workshop on Virtual Reality Interactions and Physical Simulations (VRIPHYS '17)*. 39–48.

Sheldon Andrews, Kenny Erleben, and Zachary Ferguson. 2022. Contact and Friction Simulation for Computer Graphics. In *ACM SIGGRAPH 2022 Courses (SIGGRAPH '22)*. Article 2, 124 pages.

Uri M Ascher and Eddy Boxerman. 2003. On the modified conjugate gradient method in cloth simulation. *The Visual Computer* 19 (2003), 526–531.

David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 43–54.

Christopher Batty, Florence Bertails, and Robert Bridson. 2007. A Fast Variational Framework for Accurate Solid-fluid Coupling. *ACM Trans. Graph.* 26, 3, Article 100 (2007).

Haimasree Bhattacharya, Yue Gao, and Adam W. Bargteil. 2015. A Level-Set Method for Skinning Animated Particle Data. *IEEE Transactions on Visualization and Computer Graphics* 21, 3 (mar 2015), 315–327.

Robert Bridson. 2015. *Fluid Simulation for Computer Graphics*. A K Peters/CRC Press.

Robert Bridson and Matthias Müller. 2007. Fluid Simulation: SIGGRAPH 2007 Course. In *ACM SIGGRAPH 2007 Courses* (San Diego, California) *(SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 1–81.

W Briggs, V Henson, and S Mccormick. 2000. *A Multigrid Tutorial, Second Edition*. Society for Industrial and Applied Mathematics.

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

Mingchao Cai, Andy Nonaka, John B. Bell, Boyce E. Griffith, and Aleksandar Donev. 2014. Efficient Variable-Coefficient Finite-Volume Stokes Solvers. *Communications in Computational Physics* 16, 5 (2014), 1263–1297.

Jiong Chen, Florian Schäfer, Jin Huang, and Mathieu Desbrun. 2021. Multiscale Cholesky Preconditioning for Ill-Conditioned Problems. *ACM Trans. Graph.* 40, 4, Article 81 (jul 2021), 13 pages.

Nuttapong Chentanez, Bryan E. Feldman, François Labelle, James F. O'Brien, and Jonathan R. Shewchuk. 2007. Liquid Simulation on Lattice-Based Tetrahedral Meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '07)*. 219–228.

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

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

Christian Dick, Marcus Rogowsky, and Rüdiger Westermann. 2016. Solving the Fluid Pressure Poisson Equation Using Multigrid–Evaluation and Improvements. *Visualization and Computer Graphics, IEEE Transactions on* (2016).

Zdenek Dostal and Joachim Schoberl. 2005. Minimizing Quadratic Functions Subject to Bound Constraints with the Rate of Convergence and Finite Termination. *Computational Optimization and Applications* 30, 1 (2005), 23–43.

Zdenek Dostl. 2009. *Optimal Quadratic Programming Algorithms: With Applications to Variational Inequalities* (1st ed.). Springer Publishing Company, Incorporated.

Kenny Erleben. 2013. Numerical Methods for Linear Complementarity Problems in Physics-based Animation. In *ACM SIGGRAPH 2013 Courses*. Article 8, 42 pages.

Michael Ferris and Todd Munson. 2000. Complementarity Problems in GAMS and the Path Solver. *Journal of Economic Dynamics and Control* 24 (2000), 165–188.

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

Dan Gerszewski and Adam W. Bargteil. 2013. Physics-based Animation of Large-scale Splashing Liquids. *ACM Transactions on Graphics* 32, 6, Article 185 (2013), 6 pages.

Marcus J Grote and Thomas Huckle. 1997. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing* 18, 3 (1997), 838–853.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

T. Inglis, M.-L. Eckert, J. Gregson, and N. Thuerey. 2017. Primal-Dual Optimization for Fluids. *Computer Graphics Forum* 36, 8 (2017), 354–368.

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Trans. Graph.* 30, 4, Article 78 (jul 2011), 8 pages.

Doug L. James and Christopher D. Twigg. 2005. Skinning Mesh Animations. *ACM Trans. Graph.* 24, 3 (jul 2005), 399–407.

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 (2015), 51:1–51:10 pages.

Dilip Krishnan, Raanan Fattal, and Richard Szeliski. 2013. Efficient Preconditioning of Laplacian Matrices for Computer Graphics. *ACM Trans. Graph.* 32, 4, Article 142 (jul 2013), 15 pages.

J. Kružík, D. Horák, M. Čermák, L. Pospíšil, and M. Pecha. 2020. Active set expansion strategies in MPRGP algorithm. *Advances in Engineering Software* 149 (2020), 102895.

Junyu Lai, Yangang Chen, Yu Gu, Christopher Batty, and Justin W.L. Wan. 2020. Fast and Scalable Solvers for the Fluid Pressure Equations with Separating Solid Boundary Conditions. *Computer Graphics Forum* 39, 2 (2020), 23–33.

Steve Lesser, Alexey Stomakhin, Gilles Daviet, Joel Wretborn, John Edholm, Noh-Hoon Lee, Eston Schweickart, Xiao Zhai, Sean Flynn, and Andrew Moffat. 2022. Loki: A Unified Multiphysics Simulation Framework for Production. *ACM Trans. Graph.* 41, 4, Article 50 (jul 2022), 20 pages.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental Potential Contact: Intersection-and Inversion-Free, Large-Deformation Dynamics. *ACM Trans. Graph.* 39, 4, Article 49 (July 2020), 20 pages.

Yifei Li, Tao Du, Sangeetha Grama Srinivasan, Kui Wu, Bo Zhu, Eftychios Sifakis, and Wojciech Matusik. 2022. Fluidic Topology Optimization with an Anisotropic Mixture Model. *ACM Trans. Graph.*, Article 239 (nov 2022), 14 pages.

Haixiang Liu, Nathan Mitchell, Mridul Aanjaneya, and Eftychios Sifakis. 2016. A Scalable Schur-Complement Fluids Solver for Heterogeneous Compute Platforms. *ACM Trans. Graph.* 35, 6, Article 201 (nov 2016), 12 pages.

Jinyuan Liu, Zangyueyang Xian, Yuqing Zhou, Tsuyoshi Nomura, Ercan M. Dede, and Bo Zhu. 2022. A Marker-and-Cell Method for Large-Scale Flow-Based Topology Optimization on GPU. *Struct. Multidiscip. Optim.* 65, 4 (apr 2022), 18 pages.

A. McAdams, E. Sifakis, and J. 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*. 65–74.

Rahul Narain, Abhinav Golas, and Ming C. Lin. 2010. Free-flowing Granular Materials with Two-way Solid Coupling. *ACM Transactions on Graphics* 29, 6, Article 173 (2010), 10 pages.

Maxim Naumov. 2011. Parallel Solution of Sparse Triangular Linear Systems in the Preconditioned Iterative Methods on the GPU.

Maxim Naumov. 2012. Parallel Incomplete-LU and Cholesky Factorization in the Preconditioned Iterative Methods on the GPU.

Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization* (second ed.). Springer, New York, NY, USA.

Yousef Saad. 2003. Iterative Methods for Sparse Linear Systems. *Notes* 3, 2nd Edition (2003), xviii+528. arXiv:0806.3802

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.

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 (jul 2022), 18 pages.

Jonathan Richard Shewchuk. 1994. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. (August 1994).

K. Stüben. 2001. A Review of Algebraic Multigrid. *J. Comput. Appl. Math.* 128, 1–2 (mar 2001), 281–309.

Tetsuya Takahashi and Christopher Batty. 2021. FrictionalMonolith: A Monolithic Optimization-based Approach for Granular Flow with Contact-Aware Rigid-Body Coupling. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–16.

Tetsuya Takahashi and Christopher Batty. 2022. ElastoMonolith: A Monolithic Optimization-Based Liquid Solver for Contact-Aware Elastic-Solid Coupling. *ACM Trans. Graph.* 41, 6, Article 255 (nov 2022), 19 pages.

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 (2015), 13 pages.

Osamu Tatebe. 1993. The multigrid preconditioned conjugate gradient method. In *Proceedings of the Sixth Copper Mountain Conference on Multigrid Methods (1993)*. 621–634.

Ulrich Trottenberg, Cornelius W. Oosterlee, and Anton Schuller. 2000. *Multigrid*. Academic press.

Petr Vanek, Jan Mandel, and Marian Brezina. 1996. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing* 56 (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.

Daniel Weber, Johannes Mueller-Roemer, Andre Stork, and Dieter Fellner. 2015. A Cut-Cell Geometric Multigrid Poisson Solver for Fluid Simulation. *Computer Graphics Forum* 34, 2 (2015), 481–491.

Botao Wu, Zhendong Wang, and Huamin Wang. 2022. A GPU-Based Multilevel Additive Schwarz Preconditioner for Cloth and Deformable Body Simulation. *ACM Trans. Graph.* 41, 4, Article 63 (jul 2022), 14 pages.

Xiang Yang and Rajat Mittal. 2017. Efficient relaxed-Jacobi smoothers for multigrid on parallel computers. *J. Comput. Phys.* 332 (2017), 135–142.

Xinxin Zhang. 2015. tbb_liquid_amgpcg. https://github.com/zhxx1987/tbb_liquid_amgpcg

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 (2010), 18 pages.

A FAS AS A LINEAR SOLVER

While FAS is a multigrid method designed to directly solve nonlinear equations [Briggs et al. 2000], it can be simplified for linear problems, as shown in Algorithm 2. This simplified approach improves the robustness and efficiency compared to the traditional standalone linear MG V-cycle solver, i.e., Algorithm 1 without active-sets and zero-initialization at line 1 (although these approaches are equivalent at the formulation level), because the smoothers can focus only on the difference from the current approximation. However, due to the change in the smoothing variables from $\mathbf{x}_c - \mathbf{x}_c^*$ to $\mathbf{x}_c$, we cannot directly clamp these variables (e.g., as done by Chentanez and Müller [2012] for non-negative constraints) nor initialize $\mathbf{x}_f$ to 0, which indicates that Algorithm 2 cannot be used to precondition iterative Krylov solvers [Tatebe 1993].

Algorithm 2 FAS-V-cycle($\mathbf{A}_f$, $\mathbf{x}_f$, $\mathbf{b}_f$)

if Coarsest
  Pre- and post-smoothing
else
  Pre-smoothing
  $\mathbf{x}_c^* = \mathbf{x}_c = \mathbf{P}^T\mathbf{x}_f$
  $\mathbf{b}_c = \mathbf{P}^T(\mathbf{b}_f - \mathbf{A}_f\mathbf{x}_f) - \mathbf{A}_c\mathbf{x}_c$
  FAS-V-cycle($\mathbf{A}_c$, $\mathbf{x}_c$, $\mathbf{b}_c$)
  $\mathbf{x}_f = \mathbf{x}_f + \mathbf{P}(\mathbf{x}_c - \mathbf{x}_c^*)$
  Post-smoothing
end if

B ILLUSTRATIVE OPERATION EXAMPLES

Considering the fine level linear preconditioning system $\mathbf{A}\mathbf{x}_f = \mathbf{b}_f$

$$\begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} & \mathbf{A}_{02} & \mathbf{A}_{03} \\ \mathbf{A}_{10} & \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} \\ \mathbf{A}_{20} & \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{30} & \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{f,0} \\ \mathbf{x}_{f,1} \\ \mathbf{x}_{f,2} \\ \mathbf{x}_{f,3} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{f,0} \\ \mathbf{b}_{f,1} \\ \mathbf{b}_{f,2} \\ \mathbf{b}_{f,3} \end{bmatrix}, \tag{14}$$

and active-set $\mathbf{a}_f$ (that constrains the last two elements) and the corresponding selection matrix $\mathbf{S}_f$

$$\mathbf{a}_f = \begin{bmatrix} \mathbf{a}_{f,0} \\ \mathbf{a}_{f,1} \\ \mathbf{a}_{f,2} \\ \mathbf{a}_{f,3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{S}_f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{15}$$

the residual $\mathbf{r}$ can be computed via (8) by

$$\begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} = \mathbf{S}_f\mathbf{b}_f - \mathbf{S}_f^T\mathbf{A}\mathbf{S}_f\mathbf{x}_f \tag{16}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{b}_{f,0} \\ \mathbf{b}_{f,1} \\ \mathbf{b}_{f,2} \\ \mathbf{b}_{f,3} \end{bmatrix} \tag{17}$$

$$- \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}^T \begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} & \mathbf{A}_{02} & \mathbf{A}_{03} \\ \mathbf{A}_{10} & \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} \\ \mathbf{A}_{20} & \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{30} & \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_{f,0} \\ \mathbf{x}_{f,1} \\ \mathbf{x}_{f,2} \\ \mathbf{x}_{f,3} \end{bmatrix}. \tag{19}$$

The smoothing operation (13) can be performed based on the residual $\mathbf{r}$, as discussed in Sec. 4.3. Considering the restriction operator $\mathbf{P}^T$ given as

$$\mathbf{P}^T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \tag{20}$$

we have

$$(\mathbf{P}^T)_0 \mathbf{a}_f = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = 0, \tag{21}$$

$$(\mathbf{P}^T)_1 \mathbf{a}_f = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = 2 > 0. \tag{22}$$

As such, the coarse active-set $\mathbf{a}_c$ is computed via (10) as

$$\begin{bmatrix} \mathbf{a}_{c,0} \\ \mathbf{a}_{c,1} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{23}$$