Computers & Graphics (2017)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

A Fast Linear Complementarity Problem Solver for Fluid Animation using High Level Algebra Interfaces for GPU Libraries

Michael Andersen[a], Sarah Niebe[a], Kenny Erleben[a,*]

[a] *Department of Computer Science, University of Copenhagen, Denmark*

ARTICLE INFO

Article history:
Received September 14, 2017

Keywords: Fluid Animation, Separating Solid Wall Boundary Conditions, Newton Method, Easy GPU Implementation

ABSTRACT

We address the task of computing solutions for a separating solid wall boundary condition model. We present a parallel, easy to implement, fluid linear complementarity problem solver. All that is needed is the implementation of linear operators, using an existing high-level sparse algebra GPU library. No low-level GPU programming is necessary. This means we can rely on the efficiency of a tried-and-tested library, requiring minimal debugging compared to writing more low level GPU kernels. The solver exploits matrix-vector products as computational building blocks. We block the matrix-vector products in a way that allows us to evaluate the products, without having to assemble the full systems. Our work shows speedup factors ranging up to two orders of magnitudes for larger grid resolutions.

1. Introduction and Previous Work

We use Linear Complementarity Problems (LCPs) to model separating solid wall boundary conditions. The separating behavior is preferable in large-scale simulations, as shown in [1], a work on primal-dual (PD) formulations using proximal operators. The PD approach relies on using a classification system to decide whether a cell should be a separating wall boundary conditions. The LCP approach presented here, does not require such a classification. The LCP boundary condition model is a recent approach, and so literature on the subject is still scarce. Our work is inspired by [2] and [3], but the model we derive is different, as we use a finite volume setting. For the scope of this paper, we will focus solely on LCP relevant work. We refer to the book by Bridson for a more general overview of methods for fluid simulation in Computer Graphics [4].

Our contribution is an easily implemented numerical method for solving the LCP model. We demonstrate our method by solving problem sizes which are not solvable by the method presented in [2], due to the scaling of the PATH solver. Alternative solvers that scale beyond the early PATH results have been presented. One is based on proximal operators [1] and the other is a quadratic programming (QP) approach [5]. The models presented in the two previous papers are slightly different in the sense that the first relies on a classification method for detecting separation and the second uses a mass field in the complementarity formulation.

In the fields of Computer Graphics and Mechanical Engineering, the incompressible Euler equations are used to model fluids [4, 6, 7]

$$\rho \frac{\partial \mathbf{u}}{\partial t} = \mathbf{f} - (\mathbf{u} \cdot \nabla)\,\mathbf{u} - \nabla p, \tag{1a}$$

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

where ρ is mass density, $\mathbf{u}$ is the velocity field, p is the pressure field and $\mathbf{f}$ is the external force density. Bound-

*Corresponding author
e-mail: andersen.michael@protonmail.com (Michael Andersen), sniebe@gmail.com (Sarah Niebe), kenny@di.ku.dk (Kenny Erleben)

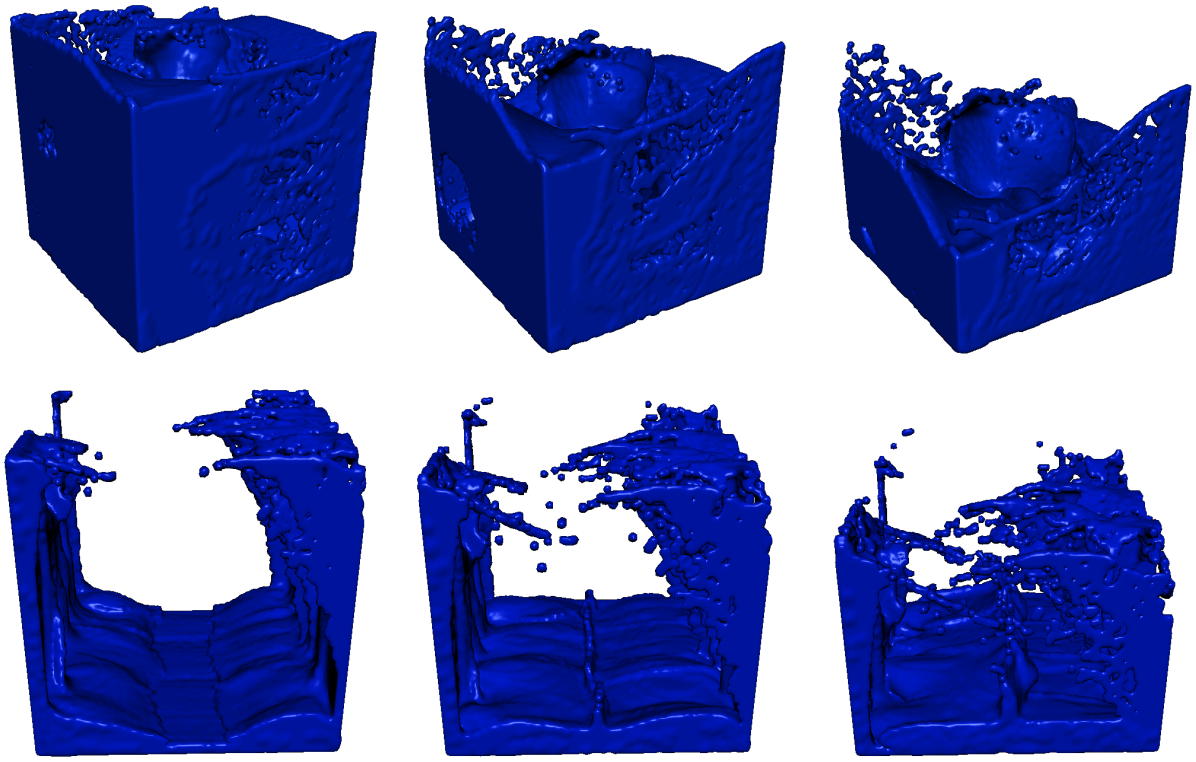


Fig. 1. Top: Frames 80, 90 and 100 of our 100^3 scene using the proposed minimum map Newton method. Bottom: Frames 140, 150 and 160 for another 100^3 scene. See supplementary video for both scenes.

ary conditions are often modeled as $p = 0$ on free surfaces between fluid and vacuum and $\mathbf{u} \cdot \mathbf{n} = 0$ between fluid and static solid walls, with outward unit normal $\mathbf{n}$. These boundary conditions are commonly referred to as *slip wall conditions*. Another commonly used boundary conditions is the *no-slip wall condition* $\mathbf{u} = \mathbf{0}$. When applying coarse computational meshes, both the slip and no-slip wall conditions tend to make the fluid stick unrealistically to the solid walls. This is illustrated in Figures 3-7.

The solid wall boundary conditions can be replaced by a different model, which allows for the fluid to separate from the walls. This leads to a new type of boundary condition that can be mathematically stated as

$$0 \leq p \quad \perp \quad \mathbf{u} \cdot \mathbf{n} \geq 0. \tag{2}$$

The notation $0 \leq x \perp y \geq 0$ means x is complementary to y. Complementarity means that when $x > 0$ then $y = 0$, and that when $y > 0$ then $x = 0$ [8]. The separating model is derived in Section 2. From a computational viewpoint, the major change is that a linear equation will be replaced by a linear complementarity problem (LCP). The LCP is much more computationally heavy and difficult to solve than the linear equation. Hence, it is computationally costly to use the separating boundary wall condition for high resolution domains. Our contribution in this work provides a computationally fast solver based on a Newton method, which we derive.

Existing LCP solvers such as PATH are generalized and do not exploit the numerical properties of the fluid problem. Our method is specialized and scales beyond previously presented work [2]. Geometric multigrid methods based on Quadratic Programming (QP) problems [9] are complex to implement, their convergence behavior is not well understood, and they are only applicable to collocated regular grids [3]. Our Newton method uses an algebraic approach and can therefore be applied to unstructured meshes. In that aspect, our approach is a more general-purpose alternative, compared to previous work. As we demonstrate in our results, our contribution is easy to apply to an existing fluid solver when the solver is based on a Preconditioned Conjugate Gradient (PCG) method for solving the pressure projection. However, in theory, we can exploit any existing Poisson sub-solver functionality. This includes, in principle, a multilevel Poisson solver resulting in a multilevel Newton method. For our implementation, we used PCG as the proof-of-concept sub-solver.

As we shall see, if we already have a fluid solver, all we need to implement is the outer Newton loop, and an appropriate line-search method. For fluid problems, our Newton method approach shows global convergence and an experimentally validated local convergence rate that supersedes the theoretical Q-linear rate of previous multi-

grid work [3].

We provide a supplementary code repository [10] containing Matlab implementations of the minimum map Newton solver, along with CUSP (CUDA/C++) based implementation. The code is meant to make validation of our work easier.

This work extends our conference paper [11] by adding investigations of the ξ-parameter for the Newton system relaxation, by documenting the volume conservation of the FLIP and LCP formulation, and giving preliminary results on using CUSP preconditioners. The framework has been extended to 3D, extra test scenes have been added, and a brief discussion of starting iterates is given.

2. Pressure Projection Formulated as a LCP

We present the ideas in the context of a single-phase flow in vacuum. The ideas extend to general multiphase flow and dynamic solid wall boundary conditions [2, 3]. Excessively coarse grids are favored in Computer Graphics to keep computational cost down. However, excessive grid coarseness presents an issue when using the solid wall boundary condition $\mathbf{u} \cdot \mathbf{n} = 0$, resulting in cell-sized thick layers of fluid sticking to the wall. This is a visually detectable, unrealistic behavior. To avoid this effect, it has been proposed to change the solid wall boundary condition to the condition stated in Equation (2), allowing the fluid to separate from the wall. Equation (2) is a complementarity condition, requiring that if $\mathbf{u} \cdot \mathbf{n} > 0$ then $p = 0$. This makes the boundary interface behave like a free surface. If, however, $p > 0$ then $\mathbf{u} \cdot \mathbf{n} = 0$ and the fluid is at rest at the wall and there must be a pressure at the wall. The complementarity boundary condition is well suited to capture the expected macroscopic fluid behavior on excessively coarse grids. However, it completely changes the mathematical problem class of the pressure solve.

Let us briefly restate a traditional pressure solver step [4]. During the last sub-step of the fractional step method, we need to solve

$$\mathbf{u}^{n+1} = \mathbf{u}' - \frac{\Delta t}{\rho}\nabla p, \tag{3a}$$

$$\nabla \cdot \mathbf{u}^{n+1} = 0, \tag{3b}$$

where $\mathbf{u}^{n+1}$ is the final divergence free velocity of the fluid and $\mathbf{u}'$ is the fluid velocity obtained from the previous step in the fractional step method. The time-step is given by Δt. Substitute (3a) in (3b) to get

$$\nabla \cdot \mathbf{u}^{n+1} = \nabla \cdot \mathbf{u}' - \frac{\Delta t}{\rho}\nabla^2 p = 0. \tag{4}$$

Introducing the spatial discretization, this results in the Poisson equation which we for notational convenience write as

$$\mathbf{A}\mathbf{p} + \mathbf{b} = \mathbf{0}, \tag{5}$$

where $\mathbf{p}$ is the vector of all cell-centered pressure values and $\mathbf{A} \equiv \left\{-\frac{\Delta t}{\rho}\nabla^2\right\}$ and $\mathbf{b} \equiv \{\nabla \cdot \mathbf{u}'\}$. Notice the SI-unit of the Poisson equation is $\left[s^{-1}\right]$. In some work the scaling $\frac{\Delta t}{\rho}$ of the $\mathbf{A}$-matrix is by linearity of the differential operator moved inside the operator, and the pressure field is redefined as $p \leftarrow \frac{\Delta t p}{\rho}$ in this case, $\mathbf{A} \equiv \left\{-\nabla^2\right\}$. Our solver works independently of the choice of the unit. The matrix $\mathbf{A}$ is a symmetric diagonal band matrix. Using low order central difference approximations in 2D, $\mathbf{A}$ will have 5 bands when using a 5-points stencil. In 3D, $\mathbf{A}$ will have 7 bands for a 7-points stencil. For regular grids, all off-diagonal bands have the same value. Further, $\mathbf{A}$ is known to be a positive semi-definite (PSD) matrix, but adding the boundary condition $\mathbf{p} = \mathbf{0}$ ensures that a unique solution can be found. Once the pressure field $\mathbf{p}$ has been determined by solving (5), it can be used to compute the last sub-step of the fractional step method (3a).

Let us revisit the complementarity condition and examine what happens if $\mathbf{u}^{n+1} \cdot \mathbf{n} > 0$ at a solid wall boundary. To start the analysis, we examine what happens with (1b) in an arbitrarily small control volume V around a solid wall boundary point,

$$\int_V \nabla \cdot \mathbf{u}^{n+1}\, dV = \oint_S \mathbf{u}^{n+1} \cdot \mathbf{n}\, dS > 0. \tag{6}$$

The last inequality follows from the assumption that $\mathbf{u}^{n+1} \cdot \mathbf{n} > 0$. Let j indicate a specific row index, and see what happens to the discrete Poisson equation corresponding to the solid wall boundary point we are looking at

$$\mathbf{A}_{j*}\mathbf{p} + \mathbf{b}_j > 0. \tag{7}$$

If on the other hand $\mathbf{u}^{n+1} \cdot \mathbf{n} = 0$ at the solid wall, we are back to

$$\mathbf{A}_{j*}\mathbf{p} + \mathbf{b}_j = 0. \tag{8}$$

Following this, we rewrite Condition (2) as the LCP

$$0 \le \mathbf{p} \quad \perp \quad \mathbf{A}\mathbf{p} + \mathbf{b} \ge 0. \tag{9}$$

Following is the derivation of the numerical contribution in this work.

3. The Minimum Map Newton Method

The core contribution of this paper is a robust, efficient and fast method for solving the LCP introduced by Equation (9). In the following we solve for $\mathbf{x} = \mathbf{p}$. That is (9) becomes

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b} \ge \mathbf{0} \tag{10a}$$

$$\mathbf{x} \ge \mathbf{0} \tag{10b}$$

$$\mathbf{x}^T\mathbf{y} = 0 \tag{10c}$$

Using the minimum map reformulation of (10) we have the root search problem where $\mathbf{H} : \mathbb{R}^n \mapsto \mathbb{R}^n$ is given by,

$$\mathbf{H}(\mathbf{x}) \equiv \begin{bmatrix} h(\mathbf{y}_1, \mathbf{x}_1) \\ \dots \\ h(\mathbf{y}_n, \mathbf{x}_n) \end{bmatrix} = \mathbf{0}. \tag{11}$$

where $h(a,b) \equiv \min(a,b)$ for $a,b \in \mathbb{R}$. Let $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$ so $\mathbf{y}_i = \mathbf{A}_{ii}\mathbf{x}_i + \mathbf{b}_i + \sum_{j\neq i}\mathbf{A}_{ij}\mathbf{x}_j$, thus

$$\mathbf{H}_i(\mathbf{x}) \equiv h(\mathbf{y}_i, \mathbf{x}_i) \tag{12a}$$

$$= \min\left(\left(\mathbf{A}_{ii}\mathbf{x}_i + \mathbf{b}_i + \sum_{j\neq i}\mathbf{A}_{ij}\mathbf{x}_j\right), \mathbf{x}_i\right), \tag{12b}$$

The basic idea is to use Newton's method to find the roots of Equation (11). Newton's method requires the derivative of $\mathbf{H}(\mathbf{x})$. Since $\mathbf{H}$ is a non-smooth function, we need to generalize the concept of a derivative [12].

Definition 3.1. *Consider any vector function $\mathbf{F} : \mathbb{R}^n \mapsto \mathbb{R}^n$, then if there exists a function $\mathbf{BF}(\mathbf{x}, \Delta\mathbf{x})$ that is* positive homogeneous *in $\Delta\mathbf{x}$, that is, for any $\alpha \geq 0$*

$$\mathbf{BF}(\mathbf{x}, \alpha\Delta\mathbf{x}) = \alpha\mathbf{BF}(\mathbf{x}, \Delta\mathbf{x}), \tag{13}$$

such that the limit

$$\lim_{\Delta\mathbf{x}\to\mathbf{0}} \frac{\mathbf{F}(\mathbf{x}+\Delta\mathbf{x}) - \mathbf{F}(\mathbf{x}) - \mathbf{BF}(\mathbf{x}, \Delta\mathbf{x})}{\| \Delta\mathbf{x} \|} = \mathbf{0} \tag{14}$$

exists. Then we say that $\mathbf{F}$ is B-differentiable at $\mathbf{x}$, and the function $\mathbf{BF}(\mathbf{x}, \cdot)$ is called the B-derivative.

The function $\mathbf{H}_i(x)$ is a *selection function* of the affine functions, $\mathbf{x}_i$ and $(\mathbf{A}\mathbf{x} + \mathbf{b})_i$. Each selection function $\mathbf{H}_i$ is Lipschitz continuous, meaning that $\mathbf{H}(\mathbf{x})$ is also Lipschitz continuous.

According to Definition 3.1, given that $\mathbf{H}(\mathbf{x})$ is Lipschitz continuous and directionally differentiable, $\mathbf{H}(\mathbf{x})$ is B-differentiable. The B-derivative $\mathbf{BH}(\mathbf{x}, \cdot)$ is continuous, piecewise linear, and positive homogeneous. Observe that the B-derivative as a function of $\mathbf{x}$ is a set-valued mapping. We will use the B-derivative to determine a descent direction for the merit function,

$$\theta(\mathbf{x}) = \frac{1}{2}\mathbf{H}(\mathbf{x})^T\mathbf{H}(\mathbf{x}). \tag{15}$$

Any minimizer of (15) is also a solution to equation (11). We use this B-derivative to formulate a linear sub-problem. The solution of this sub-problem will always provide a descent direction to (15). The largest computational task in solving the non-smooth and nonlinear system (11) is solving a large linear system of equations. A similar approach is described by [13]. The generalized Newton equation at the k^{th} iteration is

$$\mathbf{H}(\mathbf{x}^k) + \mathbf{BH}(\mathbf{x}^k, \Delta\mathbf{x}^k) = \mathbf{0}. \tag{16}$$

Each Newton iteration is finished by updating the previous iterate,

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \tau^k\Delta\mathbf{x}^k, \tag{17}$$

where τ^k is the step length and $\Delta\mathbf{x}^k$ is the Newton direction. The following theorems, see [14, 12], guarantee that $\Delta\mathbf{x}^k$ will always provide a descent direction for the merit function $\theta(\mathbf{x})$.

Theorem 3.1. *Let $\mathbf{H} : \mathbb{R}^n \mapsto \mathbb{R}^n$ be B-differentiable, and let $\theta : \mathbb{R}^n \to \mathbb{R}$ be defined by*

$$\theta(\mathbf{x}) = \frac{1}{2}\mathbf{H}(\mathbf{x})^T\mathbf{H}(\mathbf{x}). \tag{18}$$

Then θ is B-differentiable and its directional derivative at $\mathbf{x}^k$ in direction $\Delta\mathbf{x}^k$ is

$$\mathbf{B}\theta(\mathbf{x}^k, \Delta\mathbf{x}^k) = \mathbf{H}(\mathbf{x}^k)^T\mathbf{BH}(\mathbf{x}^k, \Delta\mathbf{x}^k). \tag{19}$$

Moreover, if (16) *is satisfied, the directional derivative of θ is*

$$\mathbf{B}\theta(\mathbf{x}^k, \Delta\mathbf{x}^k) = -\mathbf{H}(\mathbf{x}^k)^T\mathbf{H}(\mathbf{x}^k). \tag{20}$$

Details on proof are in [8].

Observe that a direct consequence of (20) is that any solution $\Delta\mathbf{x}^k$ of the generalized Newton equation (16) will always provide a descent direction to the merit function $\theta(\mathbf{x}^k)$. The following theorem shows that even if we solve Equation (16) approximately, we can still generate a descent direction, provided the residual is small.

Theorem 3.2. *Assume that the approximate solution $\Delta\mathbf{x}^k$ satisfies the residual equation,*

$$\mathbf{r}^k = \mathbf{H}(\mathbf{x}^k) + \mathbf{BH}(\mathbf{x}^k, \Delta\mathbf{x}^k). \tag{21}$$

Let $\theta(\mathbf{x})$ be defined by Equation (15)*. The direction $\Delta\mathbf{x}^k$ will always provide a descent direction for $\theta(\mathbf{x}^k)$ provided that*

$$\| \mathbf{H}(\mathbf{x}^k) + \mathbf{BH}(\mathbf{x}^k, \Delta\mathbf{x}^k) \| \leq \xi \ \| \mathbf{H}(\mathbf{x}^k) \|, \tag{22}$$

for some positive tolerance $\xi < 1$.

Details on proof are in [8].

We will now present an efficient way of computing the B-derivative. Given the index i we have,

$$\mathbf{H}_i(\mathbf{x}) = \begin{cases} \mathbf{y}_i & \text{if } \mathbf{y}_i < \mathbf{x}_i \\ \mathbf{x}_i & \text{if } \mathbf{y}_i \geq \mathbf{x}_i \end{cases}. \tag{23}$$

Recall that $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$. All of these are affine functions and we can compute the B-derivative $\mathbf{BH}_{ij} = \frac{\partial \mathbf{H}_i}{\partial \mathbf{x}_j}\Delta\mathbf{x}_j$ as follows [15]

(1) If $\mathbf{y}_i < \mathbf{x}_i$ then

$$\frac{\partial \mathbf{H}_i}{\partial \mathbf{x}_j} = \mathbf{A}_{ij}. \tag{24}$$

(2) If $\mathbf{y}_i \geq \mathbf{x}_i$ then

$$\frac{\partial \mathbf{H}_i}{\partial \mathbf{x}_j} = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}. \tag{25}$$

We define two index sets corresponding to our choice of active selection functions,

$$\mathcal{A} \equiv \{i \mid \mathbf{y}_i < \mathbf{x}_i\} \text{ and } \mathcal{F} \equiv \{i \mid \mathbf{y}_i \geq \mathbf{x}_i\}. \tag{26}$$

Next, we use a permutation of the indexes such that all variables with $i \in \mathcal{F}$ are shifted to the end. Hereby we create the imaginary partitioning of the B-derivative,

$$\mathbf{BH}(\mathbf{x}^k, \Delta\mathbf{x}^k) = \begin{bmatrix} \mathbf{A}_{\mathcal{A}\mathcal{A}} & \mathbf{A}_{\mathcal{A}\mathcal{F}} \\ \mathbf{0} & \mathbf{I}_{\mathcal{F}\mathcal{F}} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^k_{\mathcal{A}} \\ \Delta\mathbf{x}^k_{\mathcal{F}} \end{bmatrix}. \tag{27}$$

Notice this convenient block structure with $\mathbf{A}_{\mathcal{A}\mathcal{A}}$ being a principal submatrix of $\mathbf{A}$. The matrix $\mathbf{I}_{\mathcal{F}\mathcal{F}}$ is an identity matrix of the same dimension as the set $\mathcal{F}$.

If we use the blocked partitioning of our B-derivative from (27) then the corresponding permuted version of the Newton equation (16) is

$$\begin{bmatrix} \mathbf{A}_{\mathcal{A}\mathcal{A}} & \mathbf{A}_{\mathcal{A}\mathcal{F}} \\ \mathbf{0} & \mathbf{I}_{\mathcal{F}\mathcal{F}} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^k_{\mathcal{A}} \\ \Delta\mathbf{x}^k_{\mathcal{F}} \end{bmatrix} = - \begin{bmatrix} \mathbf{H}_{\mathcal{A}}(\mathbf{x}^k) \\ \mathbf{H}_{\mathcal{F}}(\mathbf{x}^k) \end{bmatrix}. \tag{28}$$

This can be reduced to

$$\mathbf{A}_{\mathcal{A}\mathcal{A}}\Delta\mathbf{x}^k_{\mathcal{A}} = \mathbf{A}_{\mathcal{A}\mathcal{F}}\mathbf{H}_{\mathcal{F}} - \mathbf{H}_{\mathcal{A}}. \tag{29}$$

Our problem is reduced to a potentially smaller linear system in $\Delta\mathbf{x}^k_{\mathcal{A}}$. Whether an exact solution can be found for this reduced system, depends on the matrix properties of the original matrix $\mathbf{A}$. For fluid problems, $\mathbf{A}$ is a symmetric positive semi-definite matrix, implying that the reduced matrix inherits these properties, implying that there is a risk of a singular system. As we have already shown, however, we do not need an accurate solution to guarantee a descent direction. In practice, we have found the generalized minimal residual method (GMRES) [16] to be suitable as a general-purpose choice, albeit not optimal. See Section 4 for more details on implementing sub-solvers.

To achieve better global convergence, we perform an Armijo type line-search on our merit function $\theta(\cdot)$, this is common practice in numerical optimization [17]. The ideal choice for a step length τ^k is a global minimizer of the scalar function $\psi(\tau) = \theta(\mathbf{x}_\tau)$ where $\mathbf{x}_\tau = \mathbf{x}^k + \tau\Delta\mathbf{x}^k$. In practice such a minimizer may be expensive to compute, requiring too many evaluations of $\theta(\cdot)$ and possibly $\mathbf{B}\theta(\cdot,\cdot)$. The Armijo condition stipulates that the reduction in $\psi(\tau)$ should be proportional to both the step length τ^k and the directional derivative $\nabla\psi(0) = \mathbf{B}\theta(\mathbf{x}^k, \Delta\mathbf{x}^k)$. For a sufficient decrease parameter value $\alpha \in (0, 1)$ we state this as

$$\psi(\tau^k) \leq \psi(0) + \alpha\tau^k\nabla\psi(0). \tag{30}$$

To avoid taking unacceptably short steps, we use a *backtracking* approach and terminate if τ becomes too small. Now the Armijo condition implies finding the largest $h \in \mathbf{Z}_0$ such that

$$\psi(\tau^k) \leq \psi(0) + \alpha\tau^k\nabla\psi(0), \tag{31}$$

where $\tau^k = \beta^h\tau^0$, $\tau^0 = 1$, and the step-reduction parameter $\alpha < \beta < 1$. Typical values used for α and β are: $\alpha = 10^{-4}$ and $\beta = \frac{1}{2}$ [17]. We use a projected line search to avoid getting caught in an infeasible local minima [18]. We project the line-search iterate $\mathbf{x}_\tau = \max(\mathbf{0}, \mathbf{x}^k + \tau\Delta\mathbf{x}^k)$ before computing the value of the merit function $\psi(\tau) = \theta(\mathbf{x}_\tau)$.

Algorithm 1: Projected Armijo back-tracking line-search.

Data: $\mathbf{x}^k$ and $\Delta\mathbf{x}^k$
Result: τ such that the Armijo condition is satisfied.

```
1  begin
2      (ψ_0, ∇ψ_0) ⟵ (θ(x^k), Bθ(x^k, Δx^k))
3      τ ⟵ 1
4      while Forever do
5          x_τ ⟵ max(0, x^k + τΔx^k)
6          ψ_τ ⟵ θ(x_τ)
7          if ψ_τ ≤ ψ_0 + ατ∇ψ_0 then
8              return τ
9          end
10         τ ⟵ βτ
11     end
12 end
```

Our approach is illustrated in Algorithm 1. The back-tracking line-search method we have outlined is general and could be used with any merit function. In rare cases one may experience that τ becomes too small. Thus, it may be beneficial to add an extra termination criteria after line 9 of Algorithm 1, testing whether $\tau < \delta$, where $0 < \delta \ll 1$ is a user-specified tolerance. We combine all the ingredients of the minimum map Newton method into Algorithm 2. The Newton equation can be solved using an iterative linear system method. We have had some success with the two Krylov subspace methods: PCG and GMRES. GMRES is more general than PCG and can be used for any non-singular matrix, whereas PCG requires that $\mathbf{A}$ is symmetric positive definite. PCG cannot be used for the full Newton equation, in case of the minimum map reformulation. However, for the Schur reduced system of Equation (29), PCG may be applicable if the principal submatrix is symmetric positive definite. This is the case for the specific fluid problem studied in this work.

Algorithm 2: Minimum map Newton method.

Data: $\mathbf{A}$, $\mathbf{b}$, $\mathbf{x}^0$
Result: $\mathbf{x}^k$ such the termination criteria is satisfied.

```
1  begin
2      x^k ⟵ x^0
3      repeat
4          y^k ⟵ Ax^k + b
5          H^k ⟵ min(y^k, x^k)
6          A ⟵ {i | y_i < x_i}
7          F ⟵ {i | y_i ≥ x_i}
8          solve A_AA Δx^k_A = A_AF H^k_F − H^k_A
9          τ^k ⟵ projected-line-search(...)
10         x^{k+1} ⟵ x^k + τ^k Δx^k
11         k ← k + 1
12     until x^k is converged
13 end
```

A clear benefit of using an iterative linear solver is that the full $\mathbf{A}_{\mathcal{A}\mathcal{A}}$ matrix never needs to be explicitly assembled. We only need to know the matrix-vector products, which can be evaluated directly from the finite difference schemes of the fluid solver. We exploit this to implement a fast solver, as demonstrated in Section 4.

We found that the minimum map Newton method can be started using the value $\mathbf{x}_0 = \mathbf{0}$. To increase robustness, we use a combination of termination criteria. An absolute termination criteria,

$$\theta(\mathbf{x}^{k+1}) < \varepsilon_{\mathbf{abs}}, \tag{32}$$

for some user-specified tolerance $0 < \varepsilon_{\mathbf{abs}} \ll 1$. A relative convergence test,

$$\left|\theta(\mathbf{x}^{k+1}) - \theta(\mathbf{x}^{k})\right| < \varepsilon_{\mathbf{rel}} \left|\theta(\mathbf{x}^{k})\right|, \tag{33}$$

for some user-specified tolerance $0 < \varepsilon_{\mathbf{rel}} \ll 1$. A stagnation test to identify precision issues,

$$\max_i \left|\mathbf{x}_i^{k+1} - \mathbf{x}_i^{k}\right| < \varepsilon_{\mathbf{stg}}, \tag{34}$$

for some user-specified tolerance $0 < \varepsilon_{\mathbf{stg}} \ll 1$. And lastly, a simple guard against the number of iterations exceeding a prescribed maximum to avoid infinite looping. This final criteria is needed when the Newton system is not solved to sufficient accuracy.

4. Parallel Implementation of Iterative Sub-Solvers

We now turn towards making an embarrassingly parallel implementation of our proposed minimum map Newton method. We selected CUSP as proof-of-concept for the parallelization due to its high abstraction level and low learning curve. Other alternatives exist, such as ViennaCL and cuSPARSE.

From an algorithmic viewpoint, it is preferable to solve the reduced system (29), and not the full system (28). Mainly because the reduced system will have less variables, but also because the reduced system is symmetric positive semi-definite in the specific case of the fluid problem. So in the case of the reduced system, we can apply PCG. Although the reduced equation is trivial to implement in a language such as Matlab, it is not easily done in CUSP as there is no support for index sets and indexed views of matrices and vectors. For that reason, we have opted for a different implementation strategy which we will now outline. The idea consists of having binary masks for the free and active index sets and then use element-wise multiplications to manipulate matrix-vector multiplications on the full system to equate to matrix-vector multiplications on the reduced system. This is rather technical and CUSP specific and has no implications on the algorithmic contribution of our work. However, we include the details here to facilitate reproduction of results.

Let the masks of active and free sets be defined as the binary vectors $\mathbf{a}, \mathbf{f} \in \mathbb{R}^n$ such that

$$\mathbf{a}_i = \begin{cases} 1 & \text{if } i \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases} \tag{35a}$$

$$\mathbf{f}_i = \begin{cases} 1 & \text{if } i \in \mathcal{F} \\ 0 & \text{otherwise} \end{cases} \tag{35b}$$

Notice that $\mathbf{a}^T\mathbf{f} = 0$, by definition. Also, we require strict complementarity, meaning if $\mathbf{a}_i > 0 \rightarrow \mathbf{f}_i = 0$ and $\mathbf{f}_i > 0 \rightarrow \mathbf{a}_i = 0$, but never $\mathbf{a}_i = \mathbf{f}_i = 0$. Given any mask vector $\mathbf{v}$ and a vector $\mathbf{w}$, we define $\mathbf{q} = \mathbf{v} \otimes \mathbf{w}$ as the element-wise multiplication

$$\mathbf{q}_i = \mathbf{v}_i \mathbf{w}_i \quad \forall i \in \{1, \ldots, n\} \tag{36}$$

In particular, we observe that $\mathbf{w} = \mathbf{a} \otimes \mathbf{H}$ produces a vector where $\mathbf{w}_i = 0$ if $i \in \mathcal{F}$, and $\mathbf{w}_i = \mathbf{H}_i$ if $i \in \mathcal{A}$. We now initialize the PCG solver invocation by computing a modified right-hand side, $\mathbf{q}'$, for the full system in (28)

$$\underbrace{\begin{bmatrix} \mathbf{A}_{\mathcal{A}\mathcal{A}} & \mathbf{A}_{\mathcal{A}\mathcal{F}} \\ \mathbf{0} & \mathbf{I}_{\mathcal{F}\mathcal{F}} \end{bmatrix}}_{\equiv \mathbf{M}} \underbrace{\begin{bmatrix} \Delta\mathbf{x}_{\mathcal{A}}^k \\ \Delta\mathbf{x}_{\mathcal{F}}^k \end{bmatrix}}_{\equiv \Delta\mathbf{x}} = \underbrace{-\begin{bmatrix} \mathbf{H}_{\mathcal{A}}(\mathbf{x}^k) \\ \mathbf{H}_{\mathcal{F}}(\mathbf{x}^k) \end{bmatrix}}_{\equiv \mathbf{q}}. \tag{37}$$

The modification accounts for right-hand side changes in (29),

$$\mathbf{q}' \equiv \mathbf{a} \otimes (\mathbf{A}\,(\mathbf{f} \otimes \mathbf{H})) - \mathbf{H}. \tag{38}$$

This is shown in CUSP code here

```
// v ← [0, −H_F] = [0, q_F] = F .* q
cusp::blas::xmy(free_mask, q, v);
// w ← A[0, q_F] = A v
cusp::multiply(A, v, w);
// v ← [−A_AF H_F, 0] = A .* w
cusp::blas::xmy(active_mask, w, v);
// w ← [−H_A, 0] = [q_A, 0] = A .* q
cusp::blas::xmy(active_mask, q, w);
// w ← [q'_A, 0] = [A_AF H_F − H_A, 0] = w − v
cusp::blas::axpy(v, w, -1);
```

Listing 1. The initialization of the matrix-vector product operator, creating a "virtual" Schur complement $\mathbf{q}'_{\mathcal{A}} = -\mathbf{H}_{\mathcal{A}} - (-\mathbf{A}_{\mathcal{A}\mathcal{F}}\mathbf{H}_{\mathcal{F}})$. This is used when solving for $\Delta\mathbf{x}^k$ with PCG.

Next we need to make sure that the matrix-vector product operator used by the iterative method in PCG equals the result of the reduced system, even though we are working on a full system. First we define the matrix-vector product operator as

$$\mathbf{M}'\,\Delta\mathbf{x} \equiv \mathbf{a} \otimes (\mathbf{A}\,(\mathbf{a} \otimes \Delta\mathbf{x})) - \mathbf{f} \otimes \mathbf{H} \tag{39}$$

Now we can solve the reduced system by passing the $\mathbf{M}'\Delta\mathbf{x}$ operator and $\mathbf{q}'$ vector to the PCG solver. Observe that using the operator and modified right-hand side, we do not need to actually assemble the reduced system. The drawback is that we have to use extra storage for keeping the modified right-hand side vector and for keeping temporaries when evaluating sub terms of the linear operator. The equivalent CUSP code is shown here

```
// v ← [Δx_A, 0]
cusp::blas::xmy(active_mask, dx, v);
// w ← A Δx_A
cusp::multiply(A, v, w);
// w ← [A_AA Δx_A, 0] = w_A = A .* w
cusp::blas::xmy(active_mask, w, w);
// v ← [0, −H_F] = F .* q
cusp::blas::xmy(free_mask, q, v);
// w ← [A_AA Δx_A, −H_F] = v + w
cusp::blas::axpy(v, w, 1);
```

Listing 2. Short presentation of the inner works of the linear matrix-vector product operator $\mathbf{M}'\,\Delta\mathbf{x}$ used when solving through PCG.

Usually for fluid problems, an incomplete Cholesky preconditioner is used. Although the preconditioner costs extra computation, it can often reduce the number of needed PCG iterations by two orders of magnitude. A preconditioner is essentially a matrix/linear operator $\mathbf{P}$, such that $\mathbf{P} \approx \mathbf{A}^{-1}$. When used in connection with PCG, this can be thought of as solving a left preconditioned system like

$$\mathbf{P}\,\mathbf{A}\,\mathbf{x} = \mathbf{P}\,\mathbf{b} \tag{40}$$

Clearly if we know $\mathbf{P}$, then a left preconditioner for our reduced system is given by

$$\mathbf{P}_{\mathcal{A}\mathcal{A}}\,\mathbf{A}_{\mathcal{A}\mathcal{A}}\,\Delta\mathbf{x}^k_{\mathcal{A}} = \mathbf{P}_{\mathcal{A}\mathcal{A}}\,(\mathbf{A}_{\mathcal{A}\mathcal{F}}\,\mathbf{H}_{\mathcal{F}} - \mathbf{H}_{\mathcal{A}}) \tag{41}$$

Hence, a modified preconditioner can be passed to PCG as a linear operator that will compute

$$\mathbf{P}'\,(\mathbf{r}) \equiv \mathbf{a} \otimes (\mathbf{P}\,(\mathbf{a} \otimes \mathbf{r})) + \mathbf{f} \otimes \mathbf{r} \tag{42}$$

for some vector $\mathbf{r}$ only known internally by PCG. With all our operators in place, we observe that the full modified system actually solved by PCG – written using the imaginary partitioning – is

$$\begin{bmatrix} \mathbf{A}_{\mathcal{A}\mathcal{A}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{\mathcal{F}\mathcal{F}} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^k_{\mathcal{A}} \\ \Delta\mathbf{x}^k_{\mathcal{F}} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\mathcal{A}\mathcal{F}}\,\mathbf{H}_{\mathcal{F}}(\mathbf{x}^k) - \mathbf{H}_{\mathcal{A}}(\mathbf{x}^k) \\ -\mathbf{H}_{\mathcal{F}}(\mathbf{x}^k) \end{bmatrix} \tag{43}$$

The corresponding left preconditioner is given by

$$\begin{bmatrix} \mathbf{P}_{\mathcal{A}\mathcal{A}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{\mathcal{F}\mathcal{F}} \end{bmatrix} \tag{44}$$

Hence, the parallel operator for evaluating $\mathbf{P}'\,\mathbf{r}$ is given by

$$\mathbf{P}'\,\mathbf{r} \equiv \mathcal{A} \otimes (\mathbf{P}\,(\mathcal{A} \otimes \mathbf{r})) + \mathcal{F} \otimes \mathbf{r} \tag{45}$$

The CUSP implementation is very similar to the $\mathbf{M}'\,\Delta\mathbf{x}$-operator, so we omit it. Of course, neither (43) nor (44) are ever assembled, instead we apply the linear operators as outlined above. This approach requires certain assumptions, such that the preconditioner for $\mathbf{A}$ can be reused for each Newton iteration, rather than rebuilding a preconditioner for each Newton iteration.

For the line-search method in Algorithm 1, the directional derivative of ψ is needed for the sufficient decrease test. Part of this evaluation involves the B-derivative of the minimum map reformulation $\mathbf{H}$. This can be evaluated using the same principles as for the linear sub system solver. This is shown in below.

```
// v ← AΔx
cusp::multiply(A, dx, v);
// v ← [v_A, 0] = A .* v
cusp::blas::xmy(active_mask, v, v);
// w ← [0, dx_F] = F .* Δx
cusp::blas::xmy(free_mask, dx, w);
// v ← BH(x^k, Δx^k) = [v_A, 0] + [0, dx_F] = v + w
cusp::blas::axpy(w, v, 1);
```

Listing 3. Short presentation of the inner works of the B-derivative operator used when evaluating $\mathbf{BH}(.,.)$.

5. Extension to Mixed Linear Complementarity Problems

In Section 3 we outlined a generic LCP solver. However, revisiting the ideas of Section 2 we observe that it is only on the solid wall boundary conditions that we need to solve an LCP. In the interior of the fluid domain, we have a linear system. This means we are solving a mixed LCP (MLCP). The presented LCP solver is easily adapted to solving the full fluid MLCP. Let the index set, $\mathcal{S}$, of solid wall boundary pressure nodes be

$$\mathcal{S} \equiv \{i \mid i \text{ is a solid wall boundary cell}\} \tag{46}$$

and redefine the active and non-active index sets as

$$\mathcal{A} \equiv \{i \mid \mathbf{y}_i < \mathbf{x}_i\} \ \cup \mathcal{S} \text{ and } \mathcal{F} \equiv \{i \mid \mathbf{y}_i \geq \mathbf{x}_i\} \ \setminus \mathcal{S}. \tag{47}$$

Everything else remains unchanged.

6. Results, Experiments and Discussions

For all our numerical studies, we re-implemented the 2D FLIP solver used in [2]. Figure 2 illustrates how we changed the solver to include the LCP.

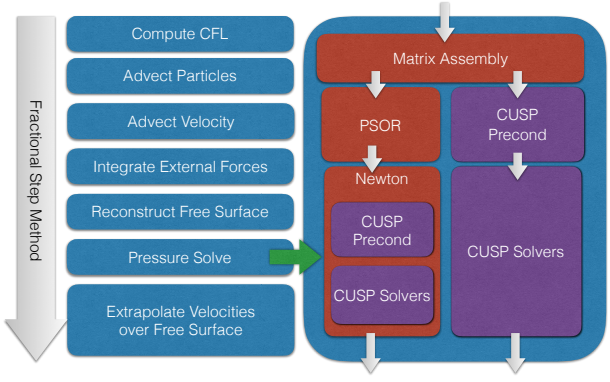


Fig. 2. Graphical illustration of how we changed the 2D fluid simulation loop from [2]. Red (CPU only) and purple (GPU accelerated) parts of the simulation loop are the ones we address in this work. We consider CUSP preconditioners: Identity, Diagonal, Bridson, Scaled Bridson and Ainv Bridson and CUSP solvers: CG, CR, GMRES, and BiCGStab.

Still frames from some of our test scenes in the supplementary movies [1] are shown in Figures 3-7 illustrating

[1] https://www.youtube.com/playlist?list=PLNtAp--NfuipGA2vXHVV60Pz2oN4xIwA0

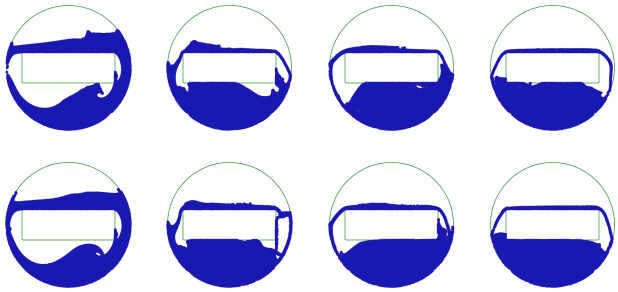


Fig. 3. Frames (30, 90, 110, 160) from the supplementary fluid simulation movie (Scene 1), comparing traditional slip conditions using a PCG solver (top row) against separating solid wall model using our minimum map Newton (bottom row).

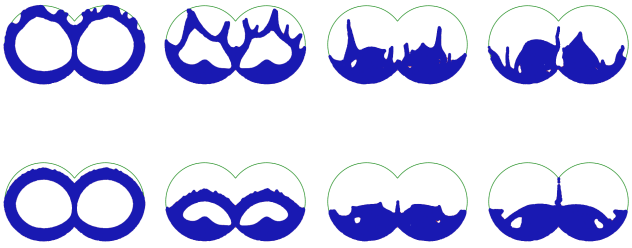


Fig. 4. Frames (10, 20, 30, 40) from the supplementary fluid simulation movie (Scene 2), comparing traditional slip conditions using a PCG solver (top row) against separating solid wall model using our minimum map Newton (bottom row).

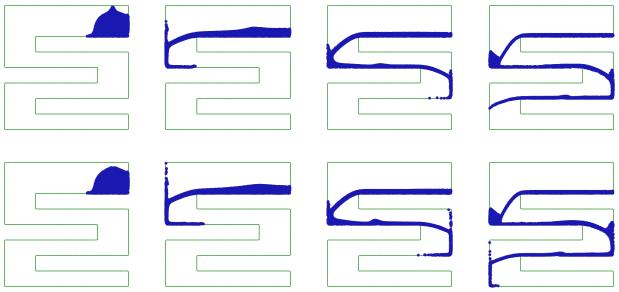


Fig. 5. Frames (10, 50, 90, 130) from the supplementary fluid simulation movie (Scene 3), comparing traditional slip conditions using a PCG solver (top row) against separating solid wall model using our minimum map Newton (bottom row).

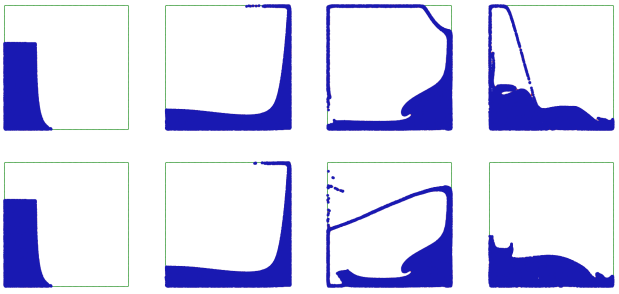


Fig. 6. Frames (10, 50, 90, 130) from the supplementary fluid simulation movie (Scene 4), comparing traditional slip conditions using a PCG solver (top row) against separating solid wall model using our minimum map Newton (bottom row).

the difference in motions. The scenes in Figures 3-7 use a grid resolution of 100×100 (10000 variables), with 20395 and 17676 liquid particles respectively. We only show the result from minimum map Newton running on a GPU device, as there was no visible difference between host CPU and GPU device. We notice in Figure 4 that when using the slip conditions, the liquid will stick to the surface around the boundaries of both circles, whereas the separating wall conditions allow the liquid to fall freely, as would be expected.

The PATH solver was used for solving the LCP of a 2D fluid simulation of relatively small sizes by [2]. The example shown in this work, appears to be a 2D 40x40 grid. We have successfully done 1024x1024 grid computations.

The work presented in [3] found the GPU accelerated LCP to be approximately 12% slower than a standard PCG solver. In our studies we found that the minimum map Newton solver to be slower than the PCG as stated in Table 1.

We found the slowdown percentage varies widely with grid resolution and the scene setup, but in general we found it to be within 5-25% range.

The work presented in [3] reports that the GPU accelerated solver uses approximately 21 msecs for 64^3 grid resolution, and 122 msecs for 128^3 grid points. Their 64^3 resolution have the same number of cells as our 2D 512^2. However, our solving times are closer to 3.5 seconds for this resolution. We believe that the large discrepancy between their and our work may be caused by us using more aggressive termination criteria. It is not clear which merit functions or termination criteria were used in [3], making one-to-one comparisons problematic.

The work presented in [1] presents a large scale 256x230x256 breaking dam example with a mean run-time per simulation time-step of 38.71 seconds. They report a 12% run-time increase, compared to solving with a regular pressure solve. The threshold values for the termination test is 10^{-3} for the PD method and 10^{-5} for the CG solver. A direct comparison of solver performance is difficult as the used hardware was not reported, nor were convergence behavior documented. Further, their method is a classification method used to identify separating solid walls, rather than letting the dynamics itself determine the proper combination. Hence, the models for separat-

Grid Resolution	Percentage
64x64	15%
128x128	25%
256x256	15%
512x512	16%
1024x1024	5%

Table 1. Slowdown of the minimum map Newton method, compared to the PCG method.

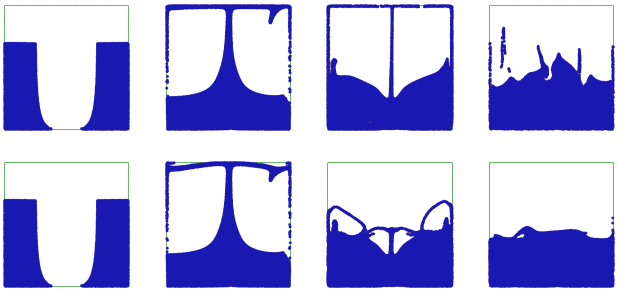


Fig. 7. Frames (10, 50, 90, 130) from the supplementary fluid simulation movie (Scene 5), comparing traditional slip conditions using a PCG solver (top row) against separating solid wall model using our minimum map Newton (bottom row).

ing solid wall boundary conditions are not the same. In the 3D test, shown in the top row of Figure 1, the pressure solve median was 46.92 seconds per update for 100^3 grid and the median for a 60^3 grid was 12.57 seconds. This was measured on a NVIDIA Tesla K40C GPU using an absolute threshold for the Newton method of 10^{-3}. For the test shown in the bottom row of Figure 1, the median took 19.73 seconds per update for a 100^3 grid using a NVIDIA GeForce GTX 1080, using the same thresholds as before. The LCP model of [5] took on average 21 seconds per frame with 11 time-steps per frame to solve with an active QP method for a $80 \times 60 \times 40$, test hardware and termination criteria are not reported. In Figure 8 we observe quadratic convergence rate of the Newton solver for the 3D example.

For the results presented here, we used an Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz and a GeForce GTX 1080. The point of our work is that one can make an easy implementation of GPU LCP solver by simply implementing the operators from Section 4 in a high-level GPU matrix library such as CUSP. Hence we compare our CUSP implementation using a CPU against using a GPU. We also compare against solving the usual slip conditions with a PCG solver to illustrate the computational tradeoff between slip-conditions and separating wall conditions.

In all our experiments, unless otherwise stated, we use the following setup of our solvers. We apply a maximum Newton iteration count of 10, and use absolute, relative and stagnation termination thresholds of 10^{-5}, and $\xi = 0.01$ (from Theorem 3.2). We use PCG as the subsolver for the minimum map Newton method, giving it a maximum iteration count of 1000 and absolute and relative termination criteria of 10^{-5}. We use the same settings when solving for slip boundary conditions with PCG. We use a maximum line-search iteration count of 100, and $\beta = 0.5$ and $\alpha = 10^{-3}$ (see Algorithm 1).

Our experiments using a preconditioner for the Newton sub-solver showed dramatic improvements in some cases. Unfortunately the experiments also showed that the overall solver fails in other cases. Hence we have omitted using preconditioning for the Newton sub-solver in our benchmarks as this do not appear to be very consistent.

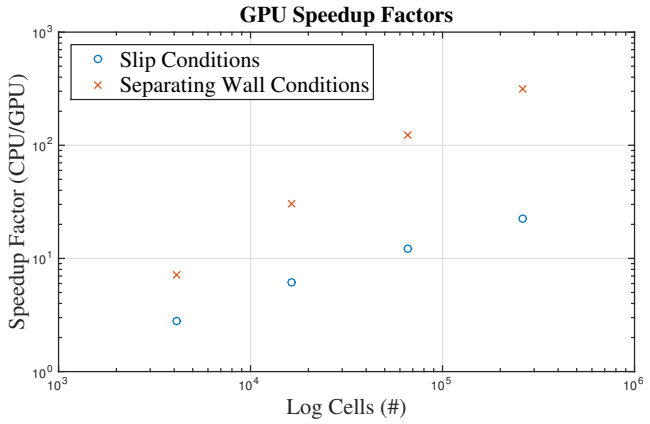


Fig. 9. Speedup factors (CPU time divided by GPU time) for increasing grid resolutions. Slip conditions are solved with PCG and separating wall conditions with minimum map Newton. Observe that GPU solver for separating wall conditions have better speedup factor.

In Figure 9 we have plotted speedup factors for both the PCG solver and the minimum map Newton solver. The speedup factors obtained for the minimum map Newton method range from low 10 and up to approximately 500 for the grid resolutions we examined. The PCG speedups are more modest in the order of 5 - 50 for the same grid resolutions. This demonstrates how implementing four operators in CUSP resulted in GPU implementation of minimum map Newton method as well as how it compares to a PCG-GPU counterpart.

A more detailed timing study would be interesting, to reveal how different parts of the solvers scale with problem size.

Figure 10 summarizes the GPU timings, divided in four groups: initialization, setup, finalization, and computation. Initialization measures time to convert to CUSP-friendly data structures. Setup includes converting to compressed sparse matrix formats (and write to device on GPU). Finalization includes converting back to fluid solver interface (and read from device on GPU). Finally, computation is the time left spent on actual computations.

In Figure 10(a) we observe that for the PCG solver initialization on GPU, converting to CUSP data structures and setting up the preconditioner is close to the actual computation time. This suggests that we cannot expect to benefit much more from the GPU for the actual computation, as initialization almost becomes the bottleneck. Further, we notice that data transfer times between CPU and GPU are not the greatest cause for concern.

In Figure 10(b) we observe that the minimum map Newton method has computation time far above the initialization phase. However, the setup and finalization on GPU (converting to/from compressed sparse matrix formats and data transfer times) are different from CPU measurements on the small-size grids, but no real difference is noticed for larger grid resolutions.

Figure 11(a) and 11(b) show the convergence rate behavior of the two solvers. Looking at the distribu-

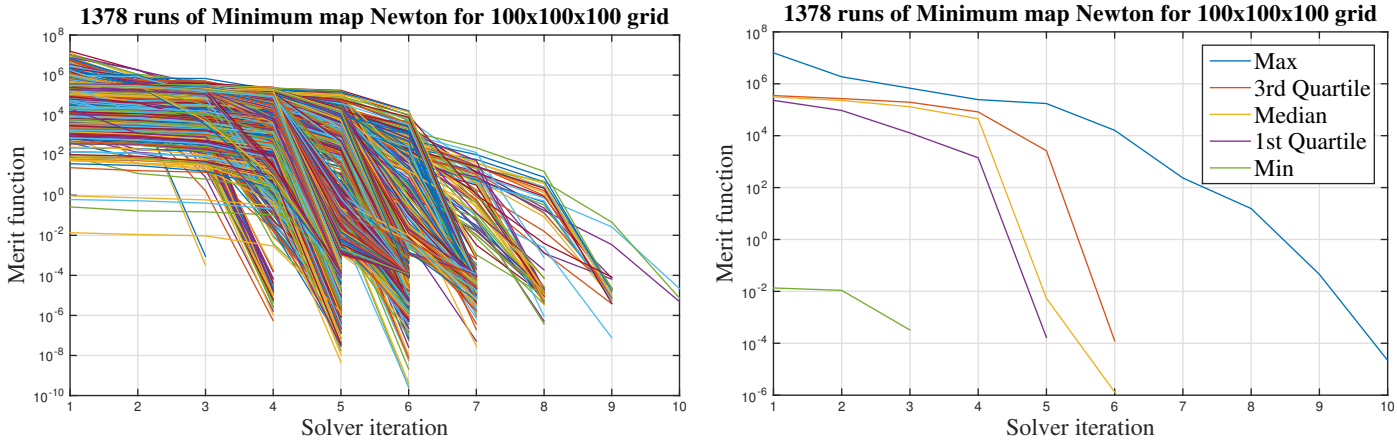


Fig. 8. On the left plot, we show convergence rate of a subset of the Newton solves for our 100^3 example. On the right plot, a quartile plot is shown of the same convergence rates. We observe quadratic convergence rates for our Newton solver in 3D.

tion of convergence plots from the minimum map Newton method we clearly see the quadratic convergence rate of the Newton method. We also observe that for a 200x200 grid, four minimum map Newton iterations work quite well for the majority of runs and six iterations appear to be an upper bound. We observe that the PCG solver has 50% of convergence rates close to its median and the remaining 50% show a large variation. In general, it appears that PCG does not make much progress after approximately 900 iterations for a 200x200 grid resolution. Note, direct comparison to PCG iterations is flawed, as each minimum map Newton iteration requires an invocation itself of a PCG solver. It is striking that 1000 PCG iterations are needed, even with the best CUSP preconditioner we could tune for (Static Scaled Bridson).

6.1. Preconditioner Results

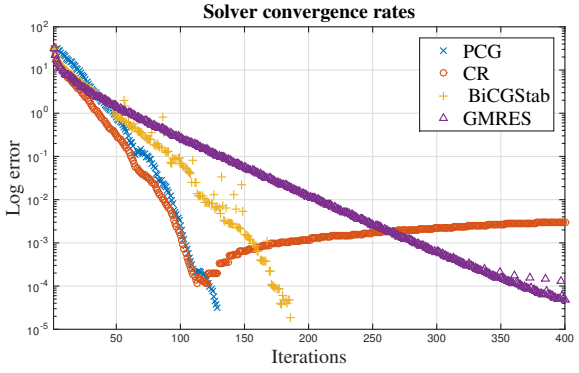
To find the most competitive CUSP solver for solving the fluid problem with slip-conditions, we examined all available CUSP combinations of Krylov subspace methods and preconditioners. We compared convergence rates from a single time-step of a fluid simulation running on a 200×200 grid resolution. Results are shown in Figure 12. Our results suggest that PCG offers better accuracy-performance tradeoff. When adding preconditioners, we observe that the "Static Scaled Bridson" preconditioner from CUSP reduces the iteration count from about 125 to 40 (close to 30%). Using this preconditioner with other available methods, clearly proves that PCG is the most suitable choice.

We repeat the experimental setup for the case of the separating wall conditions. However, not all CUSP preconditioners make sense for the minimum map Newton method. This is due to the very restrictive assumptions regarding equation (44) which lets us re-use the preconditioner in the minimum map Newton iterations. Figure 13 shows our preconditioner results. Taking into account that BiCGStab is about twice as expensive as PCG, we conclude that PCG and GMRES using an identity preconditioner are the top two choices. As PCG is more simple, and uses less memory than GMRES, we favor PCG.

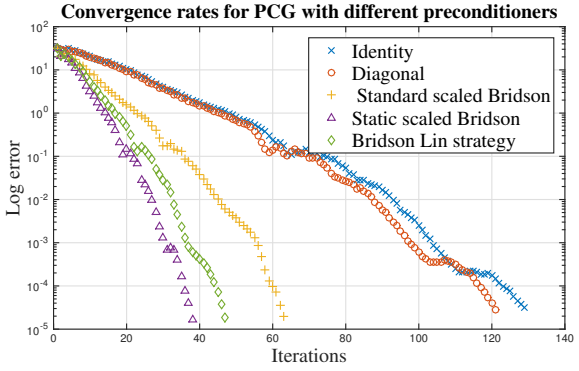
Finally, having determined the best combination of solvers and preconditions, we examine the effect of increasing grid resolution. Figure 14 presents results for both PCG and minimum map Newton method. As shown, when doubling grid resolution (roughly 4 times more variables) PCG requires about 2 times more iterations to maintain the level of accuracy. The minimum map Newton method behaves differently. The number of needed minimum map Newton iterations are 3-6, clearly fewer iterations are needed for small grids. However, the major effect seems to be when the minimum map Newton method stagnates. As our results suggest doubling the resolution, reduces accuracy from order of 10^{-6} to 10^{-3}.

6.2. Volume Conservation Studies

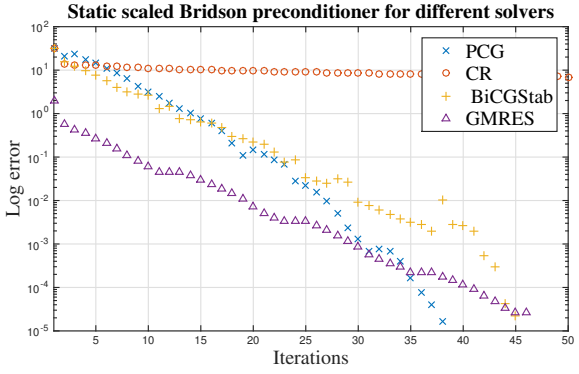
It has previously been observed that the separating solid wall boundary condition may add volume to the liquid [5]. To examine the effect in our scenes, we look at the total liquid volume over the course of 5 simulated seconds, and compare the results to the same simulations using the slip condition. Within the various scenes, we find the two conditions to result in similar qualitatively volume gain/loss plots for each individual scene. We note that scene 4 and 5 have a larger relative displacement after 1 second compared to scene 1, 2 and 3. Across the scenes, however, we see both loss and gain of volume. The results are shown in Figure 15. Scenes 1 and 2 report a general loss of volume. Scenes 3, 4, and 5 all report a gain of volume. While volume in Scene 3 is monotonously increasing till a plateau is reached, Scenes 4 and 5 start with an initial gain, followed by a loss which then plateaus at a moderate volume gain level. The results vary too greatly to say anything conclusive, apart from the fact that the two tested conditions are quantitively similar in terms of volume conservation. WE speculate that the problem originates from how the FLIP solver reconstructs the liquid surface near solid walls, as scenes with similar liquid-solid motion have similar looking volume gain profiles.



(a)



(b)



(c)

Fig. 12. A combinatorial study of solvers and preconditioners, present in the CUSP library. (a) All CUSP solvers, no preconditioner. (b) PCG method using all possible preconditioners. (c) All CUSP solvers using Static Scaled Bridson preconditioner. The study indicats that PCG using Static Scaled Bridson preconditioner is a better option.

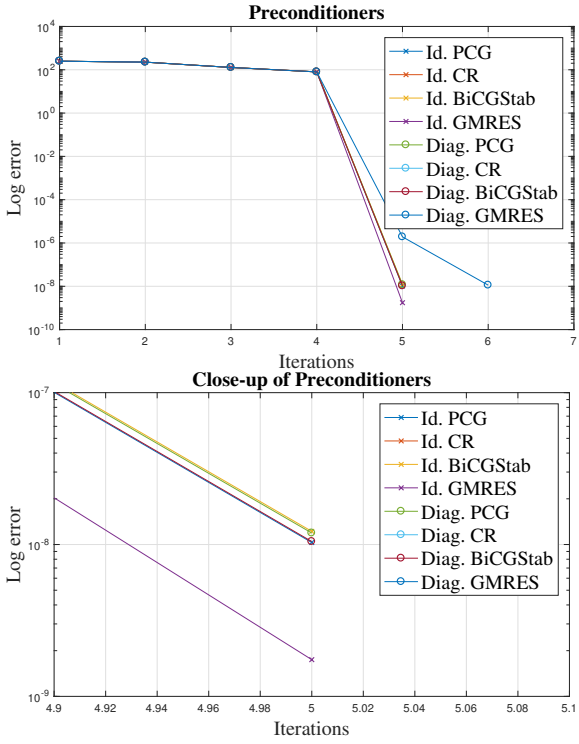
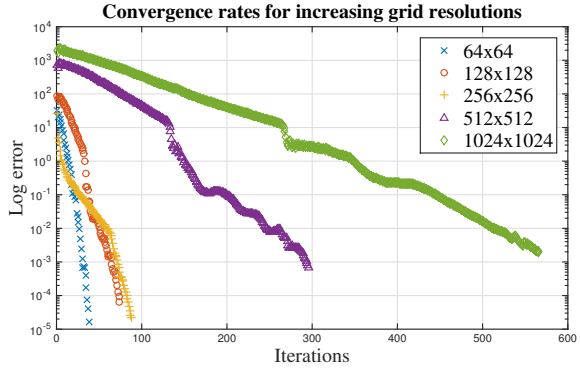
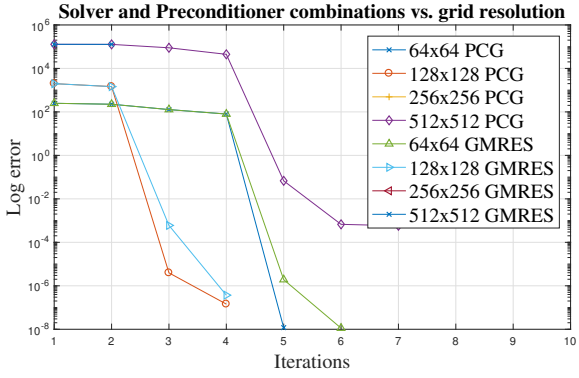


Fig. 13. Study of all applicable CUSP solver and preconditioner combinations in the minimum map Newton method. The PCG and GMRES with identity preconditioner appear most favorable.



(a) PCG using Static Scaled Bridson preconditioner



(b) Minimum map Newton using PCG and GMRES preconditioners

Fig. 14. Convergence rates as a function of grid resolution. When resolution is doubled, PCG requires twice the amount of iterates to maintain accuracy.

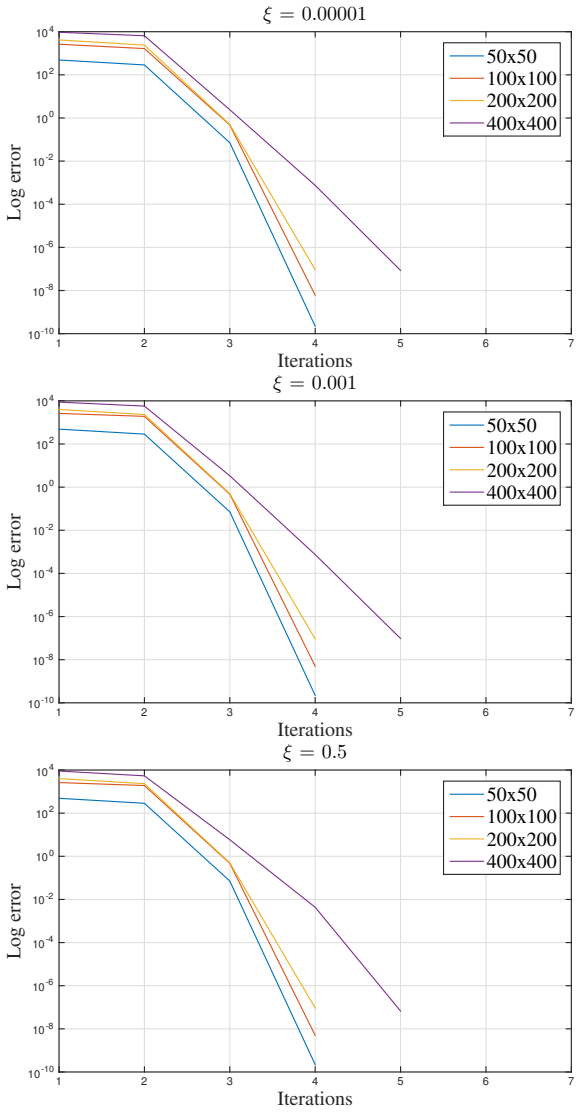


Fig. 16. Parameter study for varying ξ values, in the range $[0.00001; 0.5]$. Showing the median range of log error of the first 0.01 seconds of scene shown in Figure 4. No noticeable dependency on ξ is observed.

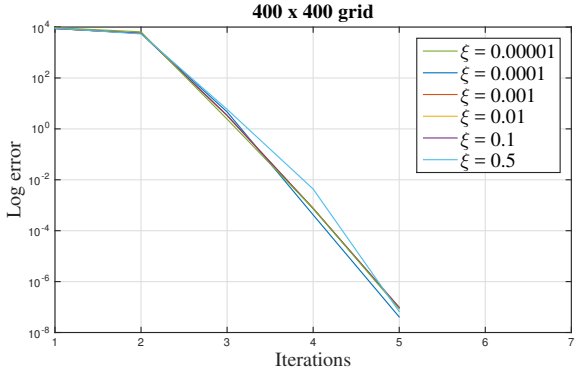


Fig. 17. More closely sampled ξ value study for a 400×400 grid. The difference suggests that for larger grid sizes a smaller ξ is preferable, but the difference is almost neglectable.

6.3. ξ Parameter Studies

We have systematically analyzed the convergence rate of the minimum map Newton method using a range of ξ-values: 0.00001, 0.0001, 0.001, 0.01, 0.1, and 0.5. The setup is Scene 2, as shown in Figure 4. The first 0.01 simulated second is analyzed, looking at the median range of the log error. A range of grid resolutions are tested for the different ξ-values, results are shown in Figure 16. Since results for grids of resolution 400×400 differ significantly from the remaining examined resolution, we did a more closely sampled study of the ξ-value for this particular resolution size, see Figure 17. Although the results seem to show that the impact of the ξ-value – at least within the range tested – is generally insignificant, there is a slight indication that for larger grid resolutions, a smaller ξ-value is preferable.

7. Conclusion and Perspective

In this work, we developed a non-smooth Newton method for separating wall boundary conditions in fluid animation.

Our experiments show clear evidence that the LCP solver is more expensive compared to similar sized fluid problems, compared to using a traditional preconditioned Conjugate Gradient solver. However, even with our limited 2D proof-of-concept visualizations, the LCP approach shows – in our opinion – very appealing visual results. The theory and solver implementation we have presented are not limited to 2D regular grid fluid simulations, it is applicable to both 3D (see Figure 1) as well as unstructured meshes. Clearly, the solver retains its convergence properties.

Our preconditioning and grid-scaling results demonstrates that CUSP provides competitive Krylov subspace solvers and preconditioners, where accuracy is not prohibited by increasing grid resolution, but the computational cost does scale with grid resolution. The minimum map Newton results suggest that quadratic convergence rate is possible. However, the results show that accuracy is lowered with increasing grid resolution. Further, it is clear that the re-usable preconditioning strategy offers poor results.

Our parameter study suggests that the ξ parameter does not influence the convergence rate of the minimum map Newton method greatly. However, there is evidence of a coupling between the ξ parameter and the grid resolution. Our modest set of resolution tests show no real sensitivity for small grid sizes. On the largest resolution test, there is a slight sensitivity. We speculate that an inverse relation exists, such that for larger grid resolutions, smaller ξ values is needed.

Our volume conservation results report interesting findings. Namely that separating solid wall boundary conditions do not always add volume to the liquid. In fact, in some cases they may cause a significant volume loss, greater than if we use the slip boundary conditions. In our opinion, this suggest that the “complementary” model

may be incomplete. The pressure-momentum equations was transformed into a LCP model, however, the transport problem that advects the density field (liquid) may need some adjustment too. We leave this modeling challenge for future work.

Computing good starting iterates is an interesting topic to pursue. Good starting iterates could dramatically improve performance on larger grids. In our experiments, we have done some initial work using PSOR and non-smooth gradient descent methods to compute starting iterates. In both cases, we use the zero-vector as input, and run a fixed number of 10 iterations. The resulting iterate we then use as the starting iterate for the minimum map Newton method. These methods are simple to implement and are consistent with the minimum map Newton method, in that they are derived from the same model and use the same merit function. However, we were not able to improve on neither performance nor robustness of the Newton method. In fact, it is far often the case that the subsolver used for solving the generalized Newton method fails completely.

Making separating solid wall boundary conditions computationally feasible could potentially open up for using even coarser grids. It would be quite interesting to validate the boundary model to see how it compares to traditional slip and no-slip wall conditions. Traditional wall boundary conditions on excessively coarse grids could suffer from too great a loss of momentum or kinetic energy. This could be overcome by the separating wall condition by allowing "bouncing" of water instead of the perfect inelastic collisions caused by the usual slip and no-slip wall conditions.

Computationally feasible complementarity problem solvers for fluid problems may hold a vast range of possible future applications and research directions. For computer animation, the LCP condition could be modified to

$$p \geq 0 \quad \perp \quad \mathbf{u} \cdot \mathbf{n} - \kappa \geq 0, \tag{48}$$

where κ could be a time-dependent user-defined "desired" divergence field to provide fluid control parameters. Non-Newtonian fluids with non-smooth dependencies between viscosity and velocity gradients may be another phenomenon that could be captured through complementarity conditions. However, the feasibility of such an application would require fast scalable solvers.

References

[1] Inglis, T, Eckert, ML, Gregson, J, Thuerey, N. Primal-dual optimization for fluids. Computer Graphics Forum 2017;.

[2] Batty, C, Bertails, F, Bridson, R. A fast variational framework for accurate solid-fluid coupling. ACM Trans Graph 2007;26.

[3] Chentanez, N, Müller, M. A multigrid fluid pressure solver handling separating solid boundary conditions. In: Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '11; New York, NY, USA: ACM. ISBN 978-1-4503-0923-3; 2011, p. 83–90.

[4] Bridson, R. Fluid Simulation for Computer Graphics. A K Peters; 2008.

[5] Gerszewski, D, Bargteil, AW. Physics-based animation of large-scale splashing liquids. ACM Trans Graph 2013;32(6):185:1–185:6.

[6] Ferziger, JH, Perić, M. Computational Methods for Fluid Dynamics. Springer; 1999.

[7] Versteeg, H, Malalasekera, W. An introduction to computational fluid dynamics: the finite volume method. Pearson Education Ltd.; 2007.

[8] Niebe, S, Erleben, K. Numerical Methods for Linear Complementarity Problems in Physics-Based Animation. Synthesis Lectures on Computer Graphics and Animation; Morgan & Claypool Publishers; 2015. ISBN 9781627053723.

[9] Mandel, J. A multilevel iterative method for symmetric, positive definite linear complementarity problems. Applied Mathematics and Optimization 1984;11(1):77–95.

[10] Erleben, K. num4lcp. Published online at `https://github.com/erleben/num4lcp`; 2011. Open source project for numerical methods for linear complementarity problems in physics-based animation.

[11] Andersen, M, Niebe, S, Erleben, K. A Fast Linear Complementarity Problem (LCP) Solver for Separating Fluid-Solid Wall Boundary Conditions. In: Jaillet, F, Zara, F, editors. Workshop on Virtual Reality Interaction and Physical Simulation. The Eurographics Association. ISBN 978-3-03868-032-1; 2017,doi:10.2312/vriphys.20171082.

[12] Pang, JS. Newton's method for b-differentiable equations. Math Oper Res 1990;15(2):311–341.

[13] Billups, SC. Algorithms for complementarity problems and generalized equations. Ph.D. thesis; University of Wisconsin at Madison; Madison, WI, USA; 1995.

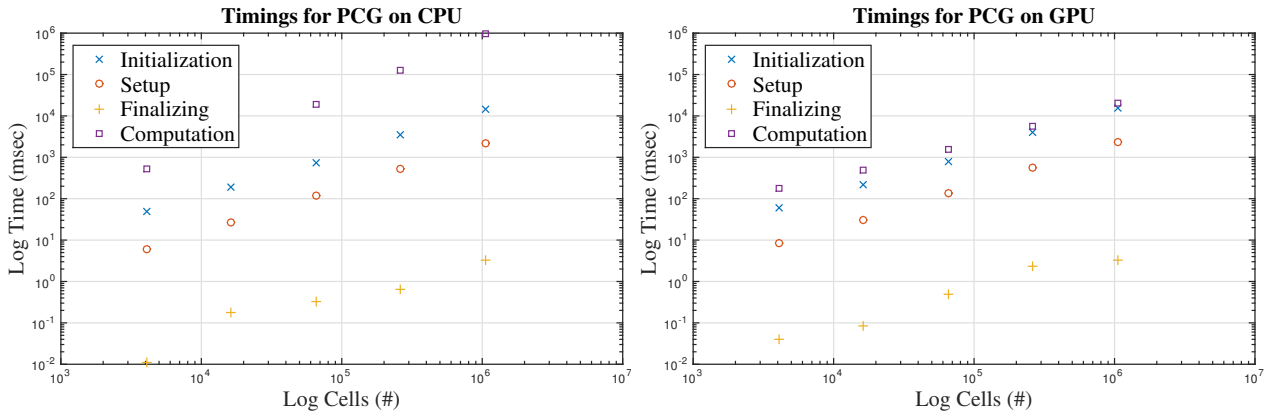
[14] Qi, L, Sun, J. A nonsmooth version of newton's method. Math Programming 1993;58(3):353–367.

[15] Scholtes, S. Introduction to piecewise differential equations; 1994. Prepring No. 53.

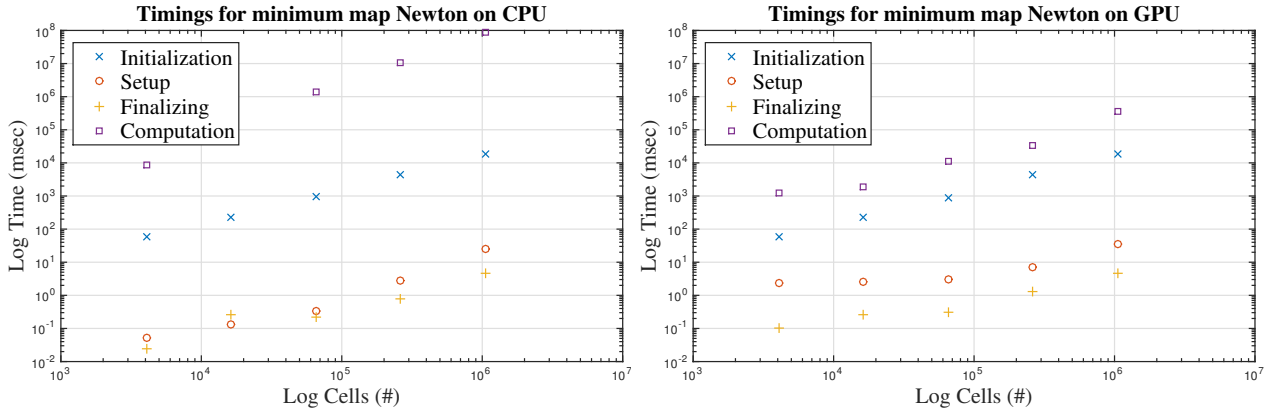
[16] Saad, Y. Iterative Methods for Sparse Linear Systems, 2nd edition. Philadelpha, PA: SIAM; 2003.

[17] Nocedal, J, Wright, SJ. Numerical optimization. Springer Series in Operations Research; New York: Springer-Verlag; 1999. ISBN 0-387-98793-2.

[18] Erleben, K, Ortiz, R. A non-smooth newton method for multibody dynamics. In: ICNAAM 2008. International conference on numerical analysis and applied mathematics 2008. 2008,.

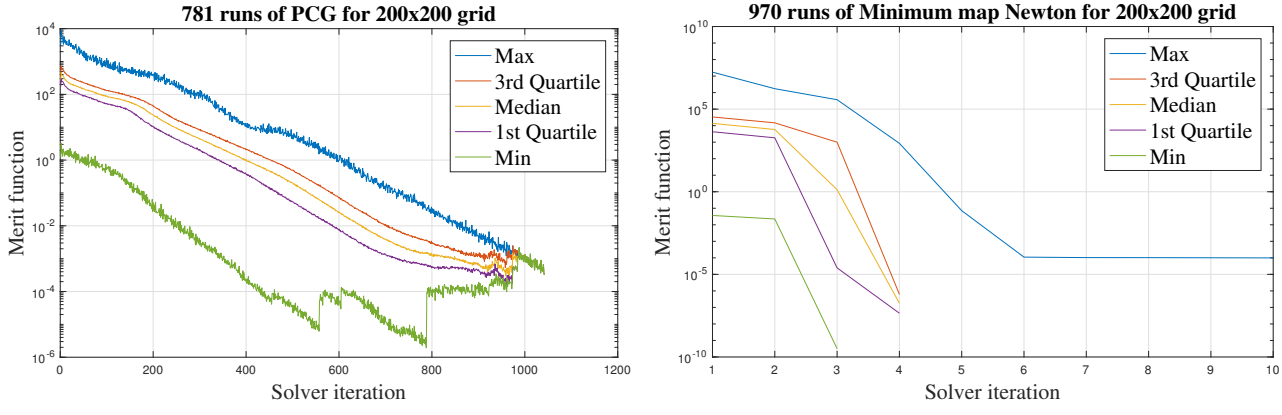


(a) PCG



(b) Minimum map Newton

Fig. 10. GPU timings for the two methods. Setup and finalization includes data transfer times and these are low compared to computation. The initialization includes converting data to GPU friendly format. For PCG the data conversion is on par with computation.



(a) PCG (b) Minimum map Newton

Fig. 11. Study of convergence rate behavior for the two methods. The distribution of convergence rates are shown using quartiles. The max and min lines show the maximum and minimum merit values and the quartiles show how 25% of the merit values are distributed with respect to the median.

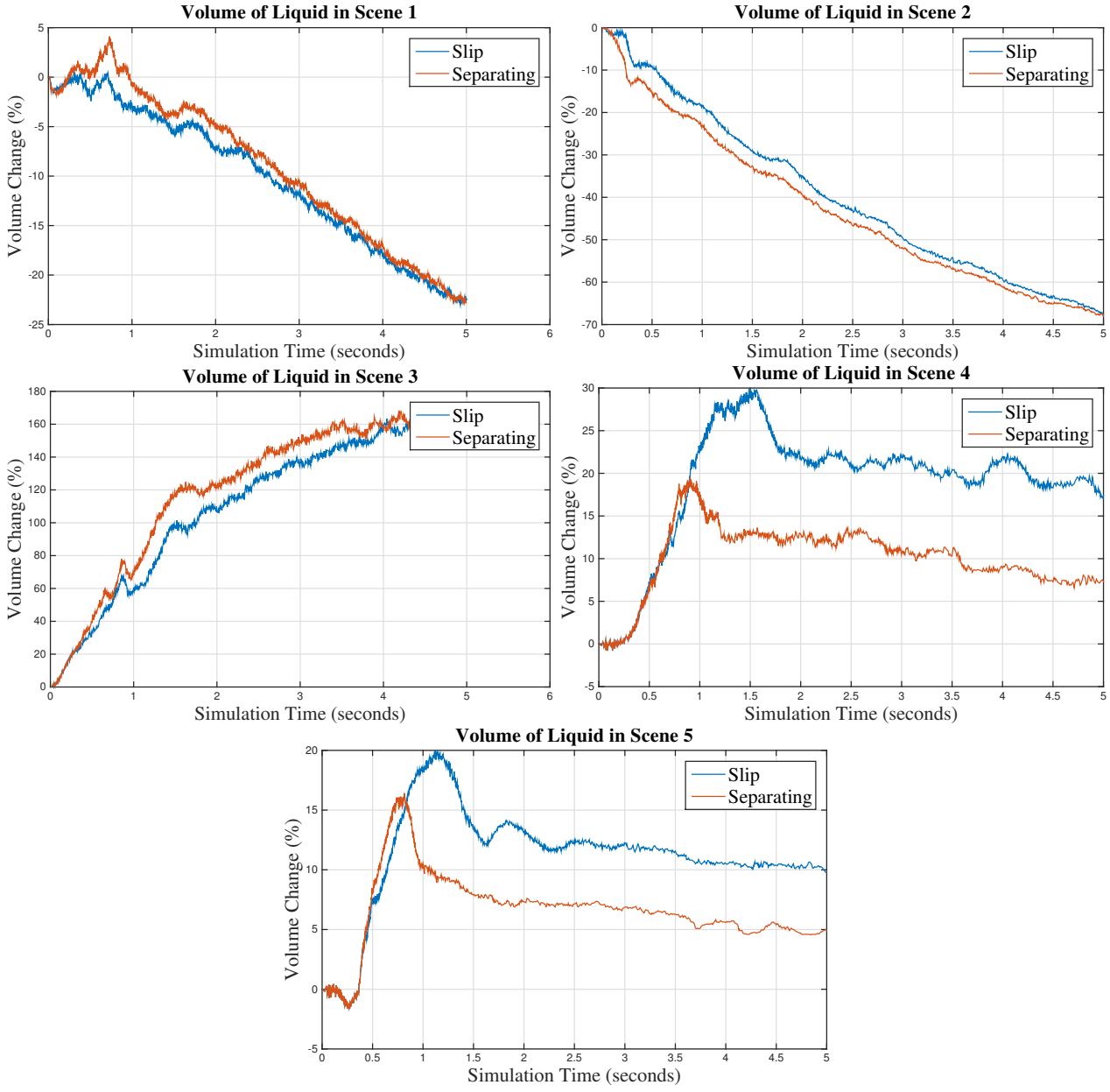


Fig. 15. Volume conservation study, five scenes run with both slip and separating solid wall boundary conditions. Observe the great variance across scenes. Further, separating wall boundary conditions sometimes add more volume and other times remove more volume than the slip conditions.