Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2017)
F. Jaillet and F. Zara (Editors)

A Fast Linear Complementarity Problem (LCP) Solver for Separating Fluid-Solid Wall Boundary Conditions

Michael Andersen, Sarah Niebe and Kenny Erleben

Department of Computer Science, University of Copenhagen, Denmark

Abstract

We address the task of computing solutions for a separating fluid-solid wall boundary condition model. We present an embarrassingly parallel, easy to implement, fluid LCP solver. We are able to use greater domain sizes than previous works have shown, due to our new solver. 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. Any iterative sub-solver can be used. Our work shows speedup factors ranging up to 500 for larger grid sizes.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—[Physically based modeling] Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—[Animation]

1. Introduction and Past Work

We use Linear Complementarity Problems (LCPs) to model separating fluid-solid wall boundary conditions. The LCP boundary condition model is a recent approach, and so literature on the subject is still scarce. Our work is inspired by Batty et al [BBB07, CM11]. However, we derive the model here differently, using 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 [Bri08].

Our contribution is an easy to implement numerical method for solving the LCP model. We demonstrate our method by solving problem sizes not possible in the work of [BBB07], due to the scaling of the PATH solver.

In the fields of Computer Graphics and Mechanical Engineering, the incompressible Euler equations are used to model fluids [FP99, Bri08, VM07]

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

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

where ρ is mass density, $\mathbf{u}$ is the velocity field, p is the pressure field and $\mathbf{f}$ is the external force density. Boundary conditions are often modelled 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 often termed *slip wall conditions*. Another often 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 Figure 2 and 3.

The fluid-solid wall boundary conditions may be replaced with a different model that allows for the fluid to separate at 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 $x = 0$ [NE15]. 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 a much more computationally heavy and difficult problem to solve than the linear equation. Hence, it is computationally infeasible 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 [BBB07]. Geometric multigrid methods based on Quadratic Programming (QP) problems [Man84] are complex to implement, convergence behaviour is not well understood, and are only applicable to collocated regular grids [CM11]. Our Newton method uses an algebraic approach and can

© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.

therefore be applied to unstructured meshes. In that aspect, our approach is a more general-purpose alternative compared to other work. As we demonstrate in our results, our contribution is easy to apply to an existing fluid solver, that is based on using a Preconditioned Conjugate Gradient (PCG) method for solving the pressure projection. However, in theory we can exploit any existing Poisson sub-solver functionality, this would in principle include a multilevel Poisson solver resulting in a Multilevel Newton method. For our implementations, we used PCG as our proof-of-concept sub-solver. If one already has a fluid solver then one is only required to implement the outer Newton loop and a proper line-search method. For fluid problems, our Newton method approach has global convergence and experimentally validated local convergence rate that supersedes theoretically Q-linear rate of previous multigrid work [CM11].

We provide a supplementary code repository [Erl11] 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.

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 [BBB07, CM11]. 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 behaviour. To combat this effect, it has been proposed to change the solid wall boundary condition to

$$0 \leq p \quad \perp \quad \mathbf{u}\cdot\mathbf{n} \geq 0. \tag{3}$$

This allows the fluid to separate from the wall. Condition (3) 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 conditions is well suited to capture the expected macroscopic fluid behaviour on excessive coarse grids. However, it completely changes the mathematical problem class of the pressure solve.

Let us briefly revisit a traditional pressure solve step [Bri08]. During the last sub-step of the fractional step method we need to solve

$$\mathbf{u}^{n+1} = \mathbf{u}' - \frac{\Delta t}{\rho}\nabla p, \tag{4a}$$

$$\nabla\cdot\mathbf{u}^{n+1} = 0, \tag{4b}$$

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 (4a) in (4b) to get

$$\nabla\cdot\mathbf{u}^{n+1} = \nabla\cdot\mathbf{u}' - \frac{\Delta t}{\rho}\nabla^2 p = 0. \tag{5}$$

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

$$\mathbf{A}\mathbf{p}+\mathbf{b}=\mathbf{0}, \tag{6}$$

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 $\mathbf{b}\equiv\left\{\nabla\cdot\mathbf{u}'\right\}$. 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 (6), it can be used to compute the last sub-step of the fractional step method (4a).

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,

$$\int_V \nabla\cdot\mathbf{u}^{n+1}\,dV = \oint_S \mathbf{u}^{n+1}\cdot\mathbf{n}\,dS > 0. \tag{7}$$

The last inequality follows from the assumption that $\mathbf{u}^{n+1}\cdot\mathbf{n}>0$. This mean that if we pick the row, let us call it j, of the discrete Poisson equation corresponding to the solid wall boundary point we are looking at

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

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

$$\mathbf{A}_{j*}\mathbf{p}+\mathbf{b}_j = 0. \tag{9}$$

Following this, we rewrite Condition (3) as the LCP

$$0 \leq \mathbf{p} \quad \perp \quad \mathbf{A}\mathbf{p}+\mathbf{b} \geq 0. \tag{10}$$

We now move on to the derivation of our 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 (10).

© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.

In the following we solve for $\mathbf{x} = \mathbf{p}$. That is (10) becomes

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b} \geq \mathbf{0} \tag{11a}$$
$$\mathbf{x} \geq \mathbf{0} \tag{11b}$$
$$\mathbf{x}^T\mathbf{y} = 0 \tag{11c}$$

Using the minimum map reformulation of (11) we have the root search problem where $\mathbf{H} : \mathbb{R}^n \mapsto \mathbb{R}^n$ is given by,

$$\mathbf{H}(\mathbf{x}) \equiv \begin{bmatrix} h(\mathbf{y}_1, \mathbf{x}_1) \\ \dots \\ h(\mathbf{y}_n, \mathbf{x}_n) \end{bmatrix} = \mathbf{0}. \tag{12}$$

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

$$\mathbf{H}_i(\mathbf{x}) \equiv h(\mathbf{y}_i, \mathbf{x}_i) \tag{13a}$$
$$= \min\left(\left(\mathbf{A}_{ii}\mathbf{x}_i + \mathbf{b}_i + \sum_{j \neq i} \mathbf{A}_{ij}\mathbf{x}_j\right), \mathbf{x}_i\right), \tag{13b}$$

The basic idea is to use Newton's method to find the roots of Equation (12). 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 [Pan90].

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 $\alpha \geq 0$

$$\mathbf{BF}(\mathbf{x}, \alpha\Delta\mathbf{x}) = \alpha\mathbf{BF}(\mathbf{x}, \Delta\mathbf{x}), \tag{14}$$

such that the limit

$$\lim_{\Delta\mathbf{x} \to \mathbf{0}} \frac{\mathbf{F}(\mathbf{x} + \Delta\mathbf{x}) - \mathbf{F}(\mathbf{x}) - \mathbf{BF}(\mathbf{x}, \Delta\mathbf{x})}{\| \Delta\mathbf{x} \|} = \mathbf{0} \tag{15}$$

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 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,

$$\theta(\mathbf{x}) = \frac{1}{2}\mathbf{H}(\mathbf{x})^T\mathbf{H}(\mathbf{x}). \tag{16}$$

Observe any minimizer of (16) is also a solution to equation (12). We use this B-derivative to formulate a linear sub-problem, the solution of this sub-problem will always provide a descent trajectory to (16). The largest computational task in solving the non-smooth and nonlinear system (12) is solving a large linear system of equations. A similar approach is described in [Bil95]. The generalized Newton equation at the k^{th} iteration is

$$\mathbf{H}(\mathbf{x}^k) + \mathbf{BH}(\mathbf{x}^k, \Delta\mathbf{x}^k) = \mathbf{0}. \tag{17}$$

Each Newton iteration is finished by updating the previous iterate,

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \tau^k\Delta\mathbf{x}^k, \tag{18}$$

where τ^k is the step length and $\Delta\mathbf{x}^k$ is the Newton direction. The following theorems, see [QS93, Pan90], 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

$$\theta(\mathbf{x}) = \frac{1}{2}\mathbf{H}(\mathbf{x})^T\mathbf{H}(\mathbf{x}). \tag{19}$$

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

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

Moreover, if (17) is satisfied, the directional derivative of θ is

$$\mathbf{B}\theta(\mathbf{x}^k, \Delta\mathbf{x}^k) = -\mathbf{H}(\mathbf{x}^k)^T\mathbf{H}(\mathbf{x}^k). \tag{21}$$

Details on proof are in [NE15].

Observe that a direct consequence of (21) is that any solution $\Delta\mathbf{x}^k$ of the generalized Newton equation (17) will always provide a descent direction to the merit function $\theta(\mathbf{x}^k)$. The following theorem shows that even if we solve Equation (17) 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,

$$\mathbf{r}^k = \mathbf{H}(\mathbf{x}^k) + \mathbf{BH}(\mathbf{x}^k, \Delta\mathbf{x}^k). \tag{22}$$

Let $\theta(\mathbf{x})$ be defined by Equation (16). The direction $\Delta\mathbf{x}^k$ will always provide a descent direction for $\theta(\mathbf{x}^k)$ provided that

$$\| \mathbf{H}(\mathbf{x}^k) + \mathbf{BH}(\mathbf{x}^k, \Delta\mathbf{x}^k) \| \leq \gamma \| \mathbf{H}(\mathbf{x}^k) \|, \tag{23}$$

for some positive tolerance $\gamma < 1$.

Details on proof are in [NE15].

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

$$\mathbf{H}_i(\mathbf{x}) = \begin{cases} \mathbf{y}_i & \text{if } \mathbf{y}_i < \mathbf{x}_i \\ \mathbf{x}_i & \text{if } \mathbf{y}_i \geq \mathbf{x}_i \end{cases}. \tag{24}$$

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 [Sch94]

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

$$\frac{\partial \mathbf{H}_i}{\partial \mathbf{x}_j} = \mathbf{A}_{ij}. \tag{25}$$

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

$$\frac{\partial \mathbf{H}_i}{\partial \mathbf{x}_j} = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}. \tag{26}$$

© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.

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

$$\mathcal{A} \equiv \{i \mid \mathbf{y}_i < \mathbf{x}_i\} \text{ and } \mathcal{F} \equiv \{i \mid \mathbf{y}_i \geq \mathbf{x}_i\}. \tag{27}$$

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,

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

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 (28) then the corresponding permuted version of the Newton equation (17) is

$$\begin{bmatrix} \mathbf{A}_{\mathcal{A}\mathcal{A}} & \mathbf{A}_{\mathcal{A}\mathcal{F}} \\ \mathbf{0} & \mathbf{I}_{\mathcal{F}\mathcal{F}} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^k_{\mathcal{A}} \\ \Delta\mathbf{x}^k_{\mathcal{F}} \end{bmatrix} = - \begin{bmatrix} \mathbf{H}_{\mathcal{A}}(\mathbf{x}^k) \\ \mathbf{H}_{\mathcal{F}}(\mathbf{x}^k) \end{bmatrix}. \tag{29}$$

Observe that this reduces to

$$\mathbf{A}_{\mathcal{A}\mathcal{A}}\Delta\mathbf{x}^k_{\mathcal{A}} = \mathbf{A}_{\mathcal{A}\mathcal{F}}\mathbf{H}_{\mathcal{F}} - \mathbf{H}_{\mathcal{A}}. \tag{30}$$

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 and one might end up with 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 GMRES to be suitable as a general-purpose choice, although 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 [NW99]. 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

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

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

$$\psi(\tau^k) \leq \psi(0) + \alpha\tau^k\nabla\psi(0), \tag{32}$$

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 $\beta = \frac{1}{2}$ [NW99]. We use a projected line search to avoid getting caught in infeasible local minima [EO08]. 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 $\psi(\tau) = \theta(\mathbf{x}_\tau)$. Our approach is illustrated in Algorithm 1. The back-tracking line search

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 $(\psi_0, \nabla\psi_0) \longleftarrow (\theta(\mathbf{x}^k), \mathbf{B}\theta(\mathbf{x}^k, \Delta\mathbf{x}^k))$
3 $\tau \longleftarrow 1$
4 **while** *Forever* **do**
5 $\mathbf{x}_\tau \longleftarrow \max(\mathbf{0}, \mathbf{x}^k + \tau\Delta\mathbf{x}^k)$
6 $\psi_\tau \longleftarrow \theta(\mathbf{x}_\tau)$
7 **if** $\psi_\tau \leq \psi_0 + \alpha\tau\nabla\psi_0$ **then**
8 **return** τ
9 **end**
10 $\tau \longleftarrow \beta\tau$
11 **end**
12 **end**

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 stop 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

Algorithm 2: Minimum map Newton method

Data: $\mathbf{A}$, $\mathbf{b}$, $\mathbf{x}^0$
Result: An $\mathbf{x}^k$ such that it satisfies the termination criteria.

1 **begin**
2 $\mathbf{x}^k \longleftarrow \mathbf{x}^0$
3 **repeat**
4 $\mathbf{y}^k \longleftarrow \mathbf{A}\mathbf{x}^k + \mathbf{b}$
5 $\mathbf{H}^k \longleftarrow \min(\mathbf{y}^k, \mathbf{x}^k)$
6 $\mathcal{A} \longleftarrow \{i \mid \mathbf{y}_i < \mathbf{x}_i\}$
7 $\mathcal{F} \longleftarrow \{i \mid \mathbf{y}_i \geq \mathbf{x}_i\}$
8 **solve** $\mathbf{A}_{\mathcal{A}\mathcal{A}}\Delta\mathbf{x}^k_{\mathcal{A}} = \mathbf{A}_{\mathcal{A}\mathcal{F}}\mathbf{H}^k_{\mathcal{F}} - \mathbf{H}^k_{\mathcal{A}}$
9 $\tau^k \longleftarrow$ **projected-line-search**$(\ldots)$
10 $\mathbf{x}^{k+1} \longleftarrow \mathbf{x}^k + \tau^k\Delta\mathbf{x}^k$
11 $k \leftarrow k+1$
12 **until** $\mathbf{x}^k$ *is converged*
13 **end**

an iterative linear system method. We have had some success with the two Krylov subspace methods: preconditioned conjugate gradient method (PCG) and generalized minimal residual method (GMRES) [Saa03]. 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 sys-

© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.

tem of Equation (30), 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.

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 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,

$$\theta(\mathbf{x}^{k+1}) < \varepsilon_{\mathbf{abs}}, \tag{33}$$

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

$$\left|\theta(\mathbf{x}^{k+1}) - \theta(\mathbf{x}^{k})\right| < \varepsilon_{\mathbf{rel}} \left|\theta(\mathbf{x}^{k})\right|, \tag{34}$$

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

$$\max_i \left|\mathbf{x}_i^{k+1} - \mathbf{x}_i^{k}\right| < \varepsilon_{\mathbf{stg}}, \tag{35}$$

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.

4. Parallel Implementation of Iterative Sub-Solvers

We now turn towards making an embarrassingly parallel implementation of our proposed 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 (30), and not the full system (29). 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 apparent matrix-vector multiplications on the full system to appear as 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

$$\mathbf{a}_i = \begin{cases} 1 & \text{if } i \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases} \tag{36a}$$

$$\mathbf{f}_i = \begin{cases} 1 & \text{if } i \in \mathcal{F} \\ 0 & \text{otherwise} \end{cases} \tag{36b}$$

Notice that by definition $\mathbf{a}^T\mathbf{f} = 0$. 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 $\mathbf{a}_i = \mathbf{f}_i = 0$. Further, 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

$$\mathbf{q}_i = \mathbf{v}_i \mathbf{w}_i \quad \forall i \in \{1, \ldots, n\} \tag{37}$$

In particular, we observe that $\mathbf{w} = \mathbf{a} \otimes \mathbf{H}$ produces a vector where for $i \in \mathcal{F}$ then $\mathbf{w}_i = 0$ and for $i \in \mathcal{A}$ then $\mathbf{w}_i = \mathbf{H}_i$. We now initialise the PCG solver invocation by computing a modified right-hand-side, $\mathbf{q}'$, for the full system in (29)

$$\underbrace{\begin{bmatrix} \mathbf{A}_{\mathcal{A}\mathcal{A}} & \mathbf{A}_{\mathcal{A}\mathcal{F}} \\ \mathbf{0} & \mathbf{I}_{\mathcal{F}\mathcal{F}} \end{bmatrix}}_{\equiv \mathbf{M}} \underbrace{\begin{bmatrix} \Delta\mathbf{x}_{\mathcal{A}}^k \\ \Delta\mathbf{x}_{\mathcal{F}}^k \end{bmatrix}}_{\equiv \Delta\mathbf{x}} = \underbrace{- \begin{bmatrix} \mathbf{H}_{\mathcal{A}}(\mathbf{x}^k) \\ \mathbf{H}_{\mathcal{F}}(\mathbf{x}^k) \end{bmatrix}}_{\equiv \mathbf{q}}. \tag{38}$$

The modification accounts for right-hand side changes in (30),

$$\mathbf{q}' \equiv \mathbf{a} \otimes (\mathbf{A}\,(\mathbf{f} \otimes \mathbf{H})) - \mathbf{H}. \tag{39}$$

This is shown in CUSP code here

```
1  // v ← [0, −H_F] = [0, q_F] = F .* q
2  cusp::blas::xmy(free_mask, q, v);
3  // w ← A[0, q_F] = A v
4  cusp::multiply(A, v, w);
5  // v ← [−A_AF H_F, 0] = A .* w
6  cusp::blas::xmy(active_mask, w, v);
7  // w ← [−H_A, 0] = [q_A, 0] = A .* q
8  cusp::blas::xmy(active_mask, q, w);
9  // w ← [q'_A, 0] = [A_AF H_F − H_A, 0] = w − v
10 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 gives the result of the reduced system, even though we are working on a full system. First we define the matrix-vector product operator as

$$\mathbf{M}'\Delta\mathbf{x} \equiv \mathbf{a} \otimes (\mathbf{A}\,(\mathbf{a} \otimes \Delta\mathbf{x})) - \mathbf{f} \otimes \mathbf{H} \tag{40}$$

Now we can solve the reduced system by passing $\mathbf{M}'\Delta\mathbf{x}$ 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 evalu-

© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.

ating sub terms of the linear operator. The equivalent CUSP code is shown here

```
1   // v ← [Δx_A, 0]
2   cusp::blas::xmy(active_mask, dx, v);
3   // w ← A Δx_A
4   cusp::multiply(A, v ,w);
5   // w ← [A_AA Δx_A, 0] = w_A = A .* w
6   cusp::blas::xmy(active_mask,w, w);
7   // v ← [0, −H_F] = F .* q
8   cusp::blas::xmy(free_mask, q, v);
9   // w ← [A_AA Δx_A, −H_F] = v + w
10  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

$$\mathbf{P}\mathbf{A}\mathbf{x} = \mathbf{P}\mathbf{b} \tag{41}$$

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

$$\mathbf{P}_{\mathcal{A}\mathcal{A}}\mathbf{A}_{\mathcal{A}\mathcal{A}}\Delta\mathbf{x}^k_{\mathcal{A}} = \mathbf{P}_{\mathcal{A}\mathcal{A}}(\mathbf{A}_{\mathcal{A}\mathcal{F}}\mathbf{H}_{\mathcal{F}} - \mathbf{H}_{\mathcal{A}}) \tag{42}$$

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

$$\mathbf{P}'(\mathbf{r}) \equiv \mathbf{a} \otimes (\mathbf{P}(\mathbf{a} \otimes \mathbf{r})) + \mathbf{f} \otimes \mathbf{r} \tag{43}$$

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

$$\begin{bmatrix} \mathbf{A}_{\mathcal{A}\mathcal{A}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{\mathcal{F}\mathcal{F}} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^k_{\mathcal{A}} \\ \Delta\mathbf{x}^k_{\mathcal{F}} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\mathcal{A}\mathcal{F}}\mathbf{H}_{\mathcal{F}}(\mathbf{x}^k) - \mathbf{H}_{\mathcal{A}}(\mathbf{x}^k) \\ -\mathbf{H}_{\mathcal{F}}(\mathbf{x}^k) \end{bmatrix} \tag{44}$$

The corresponding left preconditioner is given by

$$\begin{bmatrix} \mathbf{P}_{\mathcal{A}\mathcal{A}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{\mathcal{F}\mathcal{F}} \end{bmatrix} \tag{45}$$

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

$$\mathbf{P}'\mathbf{r} \equiv \mathcal{A} \otimes (\mathbf{P}(\mathcal{A} \otimes \mathbf{r})) + \mathcal{F} \otimes \mathbf{r} \tag{46}$$

The CUSP implementation is very similar to the $\mathbf{M}'\Delta\mathbf{x}$-operator, so we omit it. Of course, neither (44) nor (45) 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.

```
1   // v ← AΔx
2   cusp::multiply(A, dx, v);
3   // v ← [v_A, 0] = A .* v
4   cusp::blas::xmy(active_mask, v, v);
5   // w ← [0, dx_F] = F .* Δx
6   cusp::blas::xmy(free_mask, dx, w);
7   // v ← BH(x^k, Δx^k) = [v_A, 0] + [0, dx_F] = v + w
8   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 enhanced to solve the full fluid MLCP. Let the index set, $\mathcal{S}$, of solid wall boundary pressure nodes be

$$\mathcal{S} \equiv \{i \mid i \text{ is a solid wall boundary cell}\} \tag{47}$$

and redefine the active and non-active index sets as

$$\mathcal{A} \equiv \{i \mid \mathbf{y}_i < \mathbf{x}_i\} \cup \mathcal{S} \text{ and } \mathcal{F} \equiv \{i \mid \mathbf{y}_i \geq \mathbf{x}_i\} \setminus \mathcal{S}. \tag{48}$$

Everything else remains unchanged.

6. Results, Experiments and Discussions

For all our numerical studies, we re-implemented the 2D FLIP solver accompanying [BBB07]. Figure 1 illustrates how we changed the solver to include the LCP.

Still frames from some of our test scenes in the supplementary movies † are shown in Figures 2 and 3 illustrating the difference in motions. The scene in Figures 2 and 3 uses a grid resolution of 80×80 (6400 variables), with 20395 and 17676 liquid particles respectively. We only show the result from minimum map Newton running on GPU device, as there was no visible difference between host CPU and GPU device. We notice that when using the slip conditions, the liquid will stick to the surface around the boundaries of both circles, whereas the separation 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 in [BBB07]. The example

† https://www.youtube.com/playlist?list=PLNtAp--NfuipGA2vXHVV60Pz2oN4xIwA0

© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.

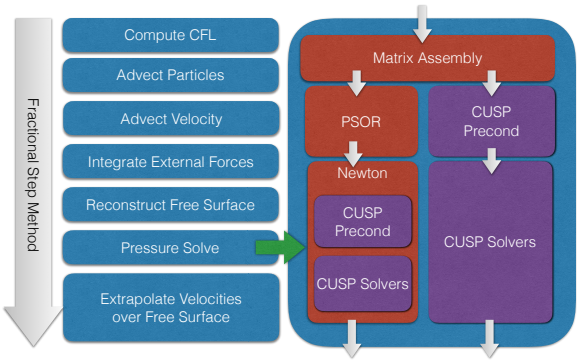


Figure 1: *Graphical illustration of how we changed the 2D fluid simulation loop from [BBB07]. Red (CPU only) and purple (GPU accelerated) parts of the simulation loop are the ones we address in this work. We consider CUSP preconditioners: Identity, Diagonal, Bridson, Scaled Bridson and Ainv Bridson and CUSP solvers: CG, CR, GMRES, and BiCGStab.*

from [BBB07] appears to be a 2D 40x40 grid. We have successfully done 1024x1024 grid computations.

In [CM11] the GPU accelerated LCP was found 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 follows

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

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

Chentanez ea [CM11] report 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 [CM11], making one-to-one comparisons problematic.

For the results presented here, we used a Intel(R) Xeon(R) CPU E5-2620 0 @ 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 we used the following setup of our solvers. We applied a maximum Newton iteration count of 10, and used absolute, relative and stagnation termination thresholds of 10^{-5}, and $\gamma = 0.01$ (from Theorem 3.2). We used PCG as the sub-solver for the Newton method giving it a maximum iteration count of 1000 and absolute and relative termination criteria of 10^{-5}. We used the same settings when solving for slip boundary conditions with PCG (more details in Appendix. We used a maximum line search iteration count of 100, and $\beta = 0.5$ and $\alpha = 10^{-3}$ (see Algorithm 1).

Our experiments with 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 these does not appear to be very consistent.

In Figure 4 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 sizes we tested. The PCG speedups are more modest in the order of 5 - 50 for the same grid sizes. 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.

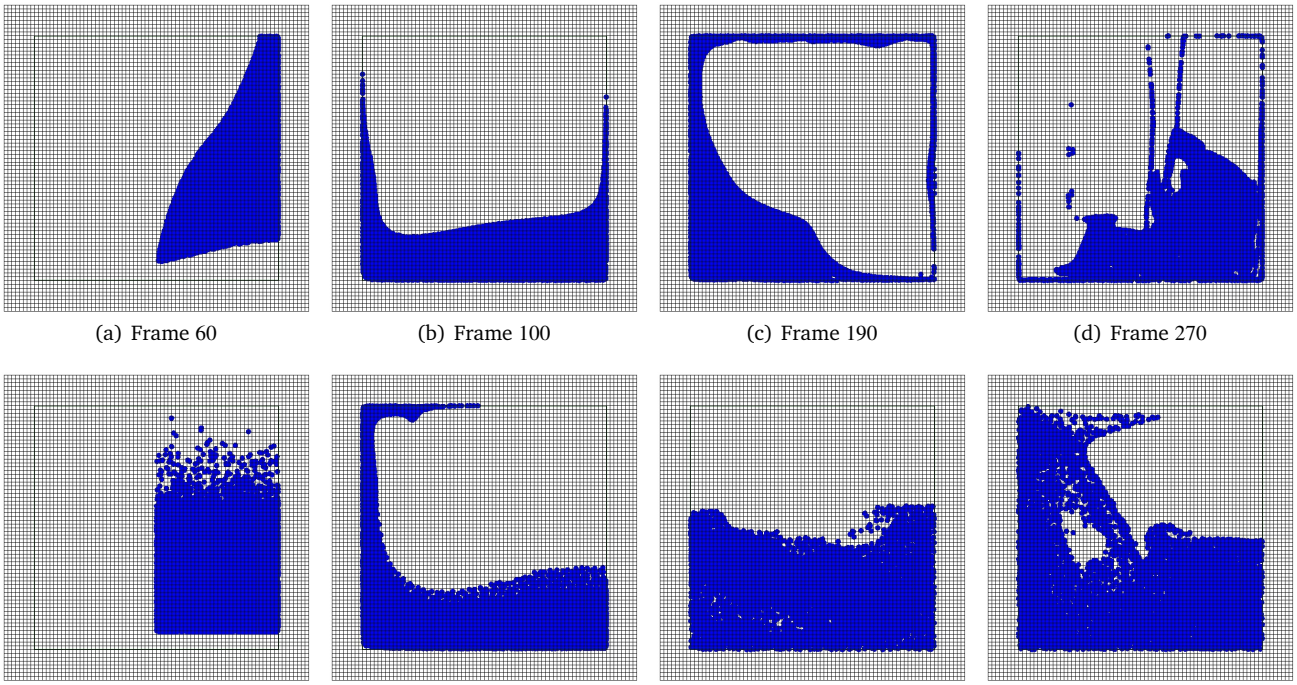
A more detailed timing study would be interesting, to reveal how different parts of the solvers scale with problem sizes.

In Figure 5 we observe that for the PCG solver initialization on GPU, converting into CUSP data structures and setting up preconditioner are close to the actual computation time. This suggests that we can not expect to benefit much more from the GPU for the actual computation, as initialization is close to become the bottleneck. Further, we notice that data transfer times between CPU and GPU are really not to be concerned with.

In Figure 6 we observe that the minimum map Newton method have 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 sizes.

Figure 7 and Figure 8 display the convergence rates of the two solvers. The Newton method clearly demonstrate the quadratic convergence rate that we hope to obtain. One should be carefull not comparing iterations directly with PCG as each Newton iteration requires an invokation itself for a PCG solver. It is striking that 1000 PCG iterations are needs even with the best CUSP preconditioner we could tune for (Static scaled Bridson).

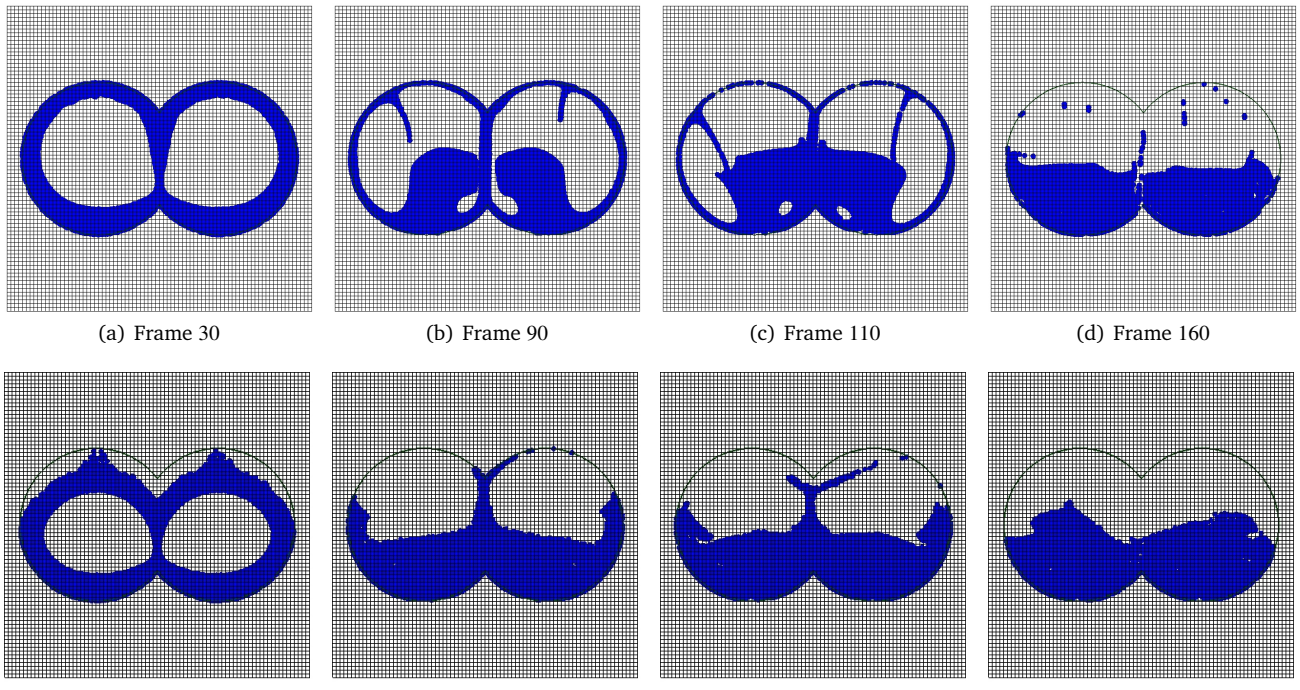
© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.



(a) Frame 60 (b) Frame 100 (c) Frame 190 (d) Frame 270

(e) Frame 60 (f) Frame 100 (g) Frame 190 (h) Frame 270

Figure 2: *Selected frames from the supplementary fluid simulation movie comparing traditional slip conditions using a PCG solver (top row) against separating solid-wall model using our minimum map Newton (bottom row).*



(a) Frame 30 (b) Frame 90 (c) Frame 110 (d) Frame 160

(e) Frame 30 (f) Frame 90 (g) Frame 110 (h) Frame 160

Figure 3: *Selected frames from the supplementary fluid simulation movie comparing traditional slip conditions using a PCG solver (top row) against separating sold-wall model using our minimum map Newton (bottom row).*

© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.

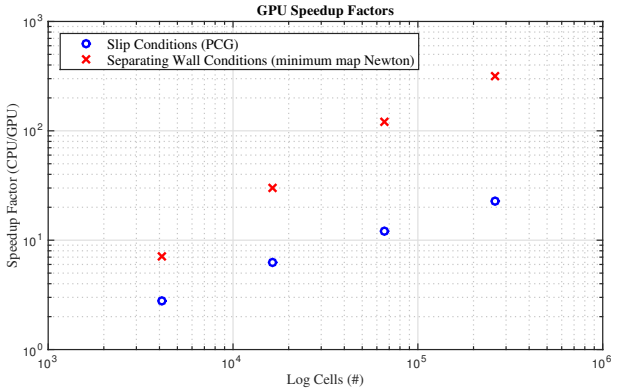


Figure 4: *Speedup factors (cpu time divided by gpu time) for increasing grid sizes. We observe an increasing speed-up with grid sizes. Clearly the minimum map Newton method benefits more from the parallelization. For a grid of size 512x512 we achieve app. 500 speedup factor.*

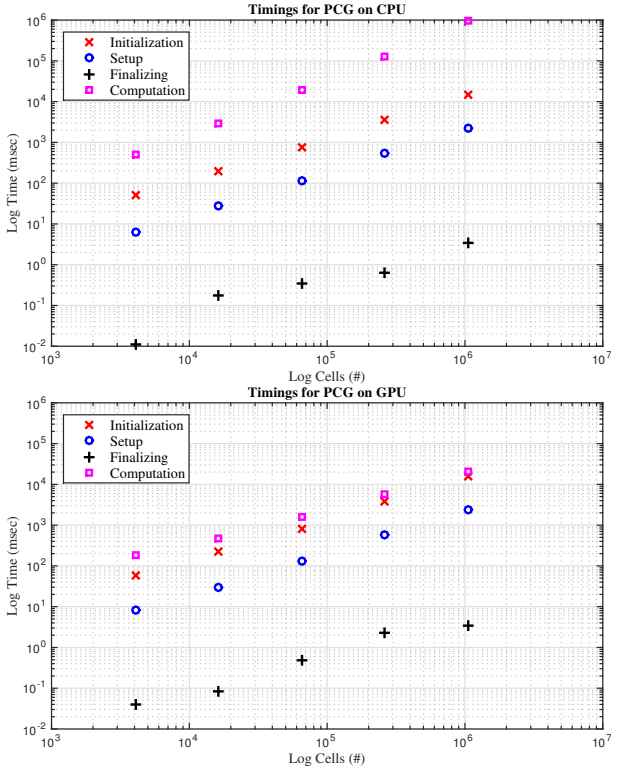


Figure 5: *Timings for the PCG solver. Initialization measures time to convert to CUSP-friendly data structures. Setup includes converting to compressed sparse matrix formats (and write to device on GPU), and finalization includes converting back to fluid solver interface (and read from device on GPU). Observe that data transfers have really no impact, and computation time is slightly lowered on GPU.*

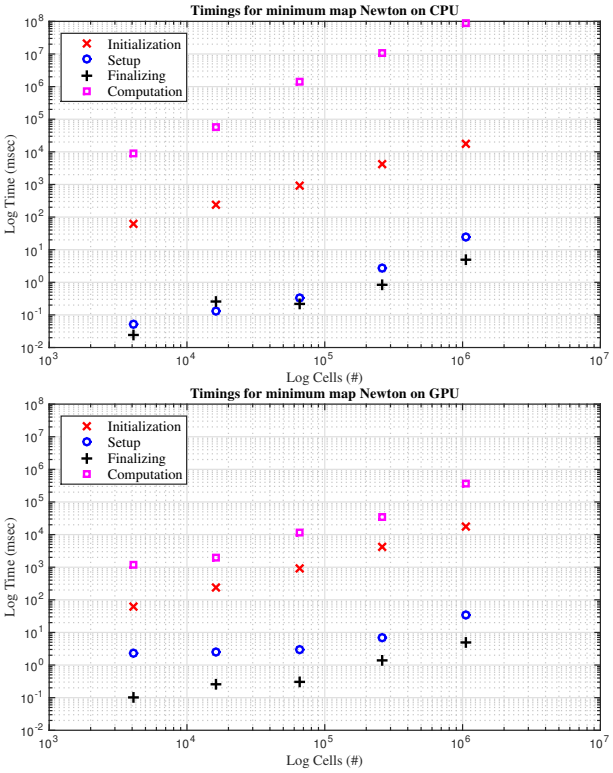


Figure 6: *Timings for minimum map Newton solver. Initialization measures time to convert to CUSP-friendly data structures. Setup includes converting to compressed sparse matrix formats (and write to device on GPU), and finalization includes converting back to fluid solver interface (and read from device on GPU). Observe that data transfers adds a small insignificant overhead on the GPU and that GPU computation times are order of magnitudes lower.*

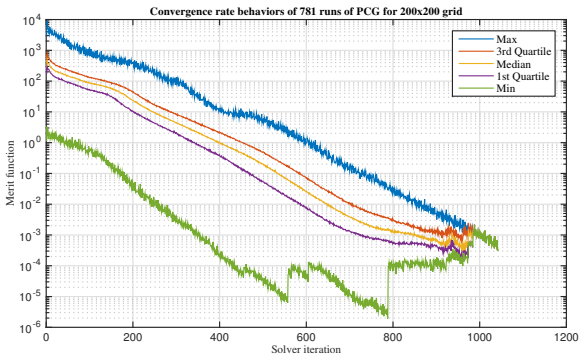


Figure 7: *We observe that the PCG solver has 50% of convergence rates close to its median and the remaining 50% are showing a large variation. In general it appears that PCG does not make much progress after approximate 900 iterations for a 200x200 grid size.*

© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.

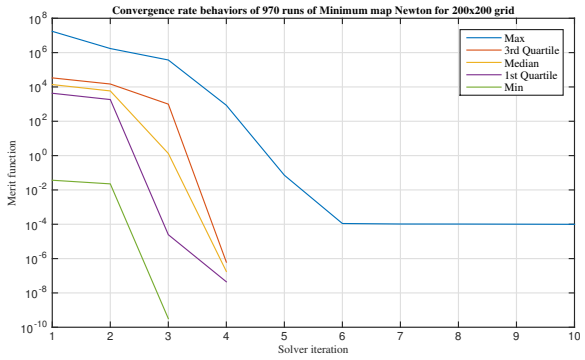


Figure 8: *Looking at the distribution of convergence plots from the minimum map Newton method we clearly see the quadratic convergence rate of the Newton method. We also observe that for a 200x200 grid 4 Newton iterations work quite well for the majority of runs and 6 iterations appear to be an upper bound.*

7. Conclusion and Perspective

In this work, we developed a non-smooth Newton method for separating wall boundary conditions in fluid animation. Detailed solutions are described for easy GPU implementation, and actual code is made freely available for other researchers.

Our experiments show clear evidence that the LCP solver is more expensive compared to similar sized fluid problems 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, and hence should be applicable to both 3D case and for unstructured meshes. Clearly, the solver retains its convergence properties. However, it will be interesting to study how the increased problem size in 3D will affect the presented solver. We leave this for future work.

Making separating fluid 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 excessive coarse grids could suffer from too large loss of momentum or kinetic energy, which the separating wall condition could overcome by allowing "bouncing" of water instead of perfect inelastic collisions 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. Obviously for computer animation one could modify the LCP condition to read $p \geq 0 \quad \perp \quad \mathbf{u} \cdot \mathbf{n} - \gamma \geq 0$ where γ 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 for others to explore this direction of work. Wall friction modelling with complementarity constraints could also be another approach to model turbulence.

References

[BBB07] BATTY C., BERTAILS F., BRIDSON R.: A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. 26* (July 2007). 1, 2, 6, 7

[Bil95] BILLUPS S. C.: *Algorithms for complementarity problems and generalized equations*. PhD thesis, University of Wisconsin at Madison, Madison, WI, USA, 1995. 3

[Bri08] BRIDSON R.: *Fluid Simulation for Computer Graphics*. A K Peters, 2008. 1, 2

[CM11] 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* (New York, NY, USA, 2011), SCA '11, ACM, pp. 83–90. 1, 2, 7

[EO08] 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). 4

[Erl11] ERLEBEN K.: num4lcp. Published online at https://github.com/erleben/num4lcp, October 2011. Open source project for numerical methods for linear complementarity problems in physics-based animation. 2

[FP99] FERZIGER J. H., PERIĆ M.: *Computational Methods for Fluid Dynamics*. Springer, 1999. 1

[Man84] MANDEL J.: A multilevel iterative method for symmetric, positive definite linear complementarity problems. *Applied Mathematics and Optimization 11*, 1 (February 1984), 77–95. 1

[NE15] 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. 1, 3

[NW99] NOCEDAL J., WRIGHT S. J.: *Numerical optimization*. Springer Series in Operations Research. Springer-Verlag, New York, 1999. 4

[Pan90] PANG J.-S.: Newton's method for b-differentiable equations. *Math. Oper. Res. 15*, 2 (1990), 311–341. 3

[QS93] QI L., SUN J.: A nonsmooth version of newton's method. *Math. Programming 58*, 3 (1993), 353–367. 3

[Saa03] SAAD Y.: *Iterative Methods for Sparse Linear Systems, 2nd edition*. SIAM, Philadelpha, PA, 2003. 4

[Sch94] SCHOLTES S.: Introduction to piecewise differential equations. Prepring No. 53, May 1994. 3

[VM07] VERSTEEG H., MALALASEKERA W.: *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education Ltd., 2007. 1

© 2017 The Author(s)
Eurographics Proceedings © 2017 The Eurographics Association.