Primal-Dual Optimization for Fluids

T. Inglis[1*], M.-L. Eckert[1*], J. Gregson[2], N. Thuerey[1]

[1]Technical University of Munich [2]University of British Columbia
[*]These authors contributed equally to this work

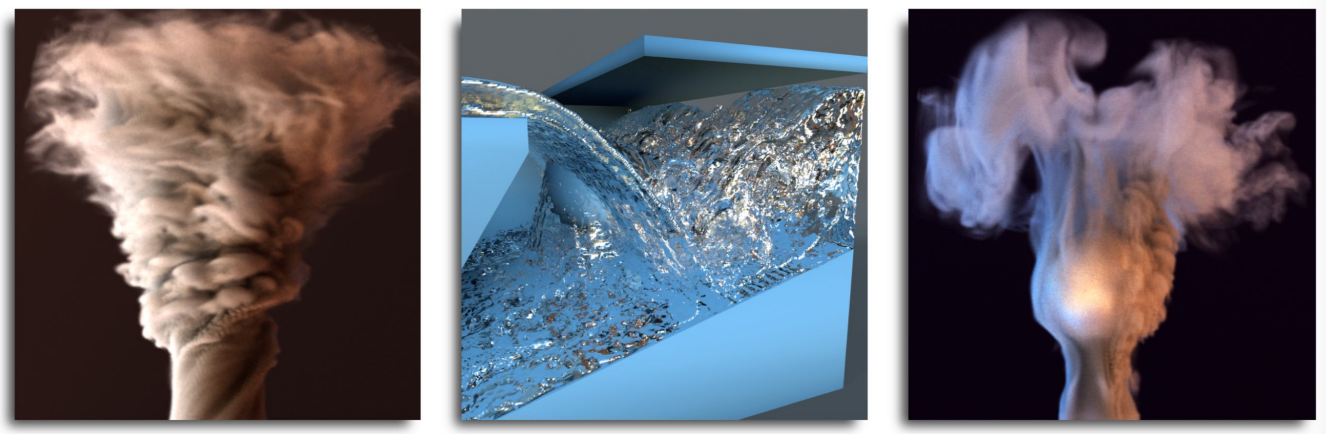


Figure 1: Our modular Primal-Dual optimization method can be applied to fluid guiding (left, right) and to simulate liquids with separating boundary conditions (center).

Abstract

We apply a novel optimization scheme from the image processing and machine learning areas, a fast Primal-Dual method, to achieve controllable and realistic fluid simulations. While our method is generally applicable to many problems in fluid simulations, we focus on the two topics of fluid guiding and separating solid-wall boundary conditions. Each problem is posed as an optimization problem and solved using our method, which contains acceleration schemes tailored to each problem. In fluid guiding, we are interested in partially guiding fluid motion to exert control while preserving fluid characteristics. With our method, we achieve explicit control over both large-scale motions and small-scale details which is valuable for many applications, such as level-of-detail adjustment (after running the coarse simulation), spatially varying guiding strength, domain modification, and resimulation with different fluid parameters. For the separating solid-wall boundary conditions problem, our method effectively eliminates unrealistic artifacts of fluid crawling up solid walls and sticking to ceilings, requiring few changes to existing implementations. We demonstrate the fast convergence of our Primal-Dual method with a variety of test cases for both model problems.

1. Introduction

Advances in fluid simulation have had a tremendous effect in engineering and graphics. Since fluids play an important role in our everyday lives, smoke and liquid simulations now routinely appear as regular elements in feature films, television and commercials. While engineering applications are primarily concerned with accuracy, the focus in graphics lies in simulating realistic behavior that can be controlled to tell a story. Therefore, it is important to provide controllable and visually plausible fluid solvers. Visual plausibility is crucial since people easily recognize unrealistic behavior due to their daily interactions with a wide range of fluid phenomena, e.g., pouring a glass of water, boiling a kettle, or driving past a chimney.

However, it is a recurring challenge to simultaneously achieve controllable and realistic behavior. Fluids are usually chaotic at human scales; miniscule perturbations regularly trigger large-scale behavior. It is almost impossible for artists to add small-scale details to a low-resolution simulation without changing the large-scale motion. Even more challenging is to modify the domain or rerun a simulation with different fluid parameters without affecting the large-scale motion. These problems can be mitigated by guiding the velocity within an optimization framework. We realize guiding

by constraining the velocity at each time step to be arbitrarily close to the current and the target velocity. At the same time, we ensure that the resulting motion is divergence-free, and we allow for spatially varying guiding parameters. This framework also benefits animators in the special effects industry who may wish to create fictitious but visually plausible fluid motion to increase entertainment value.

Common fluid solvers typically feature boundary conditions (BCs) that lead to fluid unnaturally crawling up walls and sticking to ceilings, diminishing its visual plausibility. The culprit is the solid-wall BC that enforces a normal velocity of zero at obstacle walls, thus preventing fluids from separating. While it is physically correct to leave a thin film of fluid on the wall, its thickness in simulations is on the order of the discretization and usually far too big for realistic animations. One proposed remedy by Batty et al. [BBB07] is to integrate inequality constraints into the pressure solve to allow for positive normal velocities at solid walls. Unfortunately, this greatly increases the complexity of the pressure solve of a fluid solver, and typically requires Quadratic Programming solvers. Our goal is to implement flexible separating BCs, while reusing the popular preconditioned conjugate gradient method. First, we classify boundary cells into separating and non-separating cells. We then enforce solid-wall BCs for the velocity at non-separating cells while ensuring zero divergence.

To guide simulations and realize separating BCs, we take inspiration from convex optimization. Gregson et al. [GITH14] already established a connection between fluid pressure correction and convex optimization. This connection allows us to impose all required physical constraints on the velocity for both guiding and separating BCs via an efficient alternating minimization algorithm that exploits problem structure, removing the need for a monolithic and slow framework. We introduce a novel optimization scheme from the computer vision area, the *fast*, or *first-order Primal-Dual* (PD) method proposed by Pock et al. [PCBC09]. It features an optimal convergence rate for non-smooth convex problems. Note that a whole class of primal-dual methods exists, but we will hereafter use the abbreviation *PD* to refer to this particular instance, which we will focus on due to its fast convergence properties.

Specifically, our contributions are

- the introduction of a modular convex optimization approach for fluids based on the PD method;
- a general method for handling the difficult problem of fluid guiding that involves spatially varying operators;
- a fast method to approximately invert the linear system for flow guiding in arbitrary domains; and
- a novel, practical way to handle separating BCs for liquids.

2. Previous Work

Fluid simulation has a long history in computer graphics [KM90]. One of the most widely used methods is the stable fluid solver [Sta99], for which many extensions have been proposed over the years, e.g., the grid-particle hybrid *FLIP* (Fluid Implicit Particle) method [ZB05], which is particularly popular for liquid simulations. A good overview of these methods is given in Bridson's book [Bri08]. A key component of incompressible fluid simulation is ensuring that the time-evolved velocity is divergence-free (i.e., mass-preserving), which is typically implemented via Chorin-like pressure-projections, e.g. [FF01] or [BBB07] for variational approaches.

Convex Optimization has become a powerful component of many computer vision algorithms. Even before, the works of Boyd et al. [BPC*11] have laid the foundations for many popular optimization algorithms. Closer to computer graphics, Heide et al. [HRH*13] have proposed using the PD method to correct the deficiencies of simple lenses. Recently, Heide et al. [HDN*16] have solved the difficult task of choosing the optimal image priors and optimization algorithms for image processing tasks such as deconvolution, denoising or inpainting by applying PD as well. Iterated Orthogonal Projection (IOP), the algorithm proposed by Molemaker et al. [MCPN08], is a specialized method in the convex optimization area requiring all operators to be orthogonal projections. Approaches such as position-based fluids (PBF) [MM13] also can be seen as an iterative application of a set of constraint projections. As the constraints are directly applied to the degrees of freedom, PBF is more closely related to IOP than PD, which achieves improved convergence by projecting the convex conjugate.

We extend upon the ideas presented by Gregson et al. [GITH14], where the method Alternating Direction Methods of Multipliers (ADMM) is applied, by introducing a more efficient splitting algorithm for flow guiding and BCs. Very recently, Narain et al. [NOB16] animated deformable objects with an ADMM algorithm combining a fast and robust nonlinear elasticity model with hard constraints. Our method can be seen as preconditioned version of ADMM, and its convergence is accelerated by an improved update direction for each iteration. O'Connor et al. [OV14] found ADMM outperforming PD for few iteration counts while using a restricted version of PD which does not exploit PD's full potential of optimal update directions. We demonstrate the advantages of our approach over both ADMM and IOP in Section 4.1.

Guiding A limitation of fluid simulation for computer graphics is the issue of control. Simply changing the resolution of a simulation can alter its behavior significantly. A popular class of methods circumvents this problem by synthesizing smaller scales in a decoupled fashion [BHN07, KTJG08, PTSG09]. As a consequence, the details are easy to fine-tune, but do not tightly integrate with the base flow, which is left unmodified.

Fluid guiding is a challenging example of an inverse problem for fluids. Adjoint methods have been used in engineering and graphics [GP00, MTPS04] but require differentiation of the entire fluid solver. Guide shapes [NB11] are able to add detail to free surface flows by applying velocity conditions to high-resolution simulations distant to the interface but are limited in volumetric contexts. Other approaches aiming for high-level control have proposed the use of Lagrangian coherent structures [YCZ11] or sketches [PHT*13] to give users intuitive controls. As our guiding technique takes an arbitrary flow field as input to calculate plausible and tightly coupled detail, such approaches would be a good complement for our method.

While Gregson et al. [GITH14] also demonstrate preliminary results for guided flows, their approach employed the fast Fourier transform for filtering low-frequencies and pressure-projection.

This is a very efficient approach, but it becomes impractical for more complex geometries and spatially varying guiding.

A simpler approach to guiding is via control forces [SY05b, FL04] or velocities [SY05a]. These approaches can result in artificially viscous flows, as noted by Thuerey et al. [TKRP06], who proposed a multi-scale approach based on control particles that controls low-spatial frequencies. Possibly the most similar approach to ours is the one by Nielsen et al. [NCZ*09, NC10] who use a multi-scale formulation based on low pass filters. They arrive at a monolithic system with four degrees of freedom per cell, requiring a specialized solver to reduce run time. In contrast, our control scheme decouples into separate and more easily solved subproblems via recent developments in non-smooth optimization. Furthermore, for practical purposes, we introduce an *upres* method into our workflow [KTJG08, YCZ11, HMK11, RLL*13, HK13] to facilitate the separation of low-resolution and high-resolution guiding.

Boundary Conditions play a crucial role in fluid simulation, and as such have received a significant amount of attention. Foster et al. [FF01] describe how to allow for tangential motions of liquids, and are the first to note problems with liquid unnaturally sticking to domain boundaries. Batty et al. [BBB07] propose to solve inequality constraints with Quadratic Programming, while Chentanez et al. [CMF12] observe that they can incorporate these inequalities into their multigrid solver. Methods such as the FFT-based one of R. Henderson [Hen12] are similar in spirit to our method, as they separate the BCs from the divergence-free projection step, however, without targeting separating boundaries. Other approaches realize unilateral incompressibility to allow for separation effects [NGL10], [NGCL09], [GB13] but they also require complex solvers to solve their proposed Quadratic Programming Problems. We will demonstrate how to solve for separating motions with a regular conjugate gradient (CG) solver based on our modular optimization framework.

3. Methodology

In graphics, fluids are typically simulated by solving the incompressible Euler equations, written as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} = -\nabla p + \mathbf{f}_{\text{ext}}\,, \tag{1}$$

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

where $\mathbf{u}$ is the flow velocity, p the pressure, and $\mathbf{f}_{\text{ext}}$ the external body forces. Simulations commonly proceed via operator splitting to satisfy both constraints. First, using all but the pressure term in Eq. (1), an intermediate velocity field is computed. Then a pressure projection is applied to satisfy the divergence-free condition in Eq. (2). A pressure field that exactly counteracts the divergence is computed to correct the velocity field. Orthogonality of the curl-free and divergence-free components ensures that divergence-free components of the flow are not affected and allows the pressure projection to be interpreted as an Euclidean projection onto the space of divergence-free velocity fields.

The splitting approach closely resembles convex optimization approaches originally developed for imaging inverse problems and machine learning involving non-smooth or constrained objective functions. We show how several of these approaches can be adapted to solve difficult problems in fluids.

Convex Optimization aims to solve problems of the form

$$\underset{\mathbf{x}}{\text{minimize}} \quad h(\mathbf{x}), \tag{3}$$

where h is a convex function that may be non-smooth or even discontinuous. If h is the sum of two simpler functions (say $h = f+g$), then we have

$$\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x}) + g(\mathbf{x}). \tag{4}$$

A number of recently developed algorithms target this type of problem by employing an iterative divide-and-conquer approach. These algorithms are known as *proximal methods* and are defined in terms of so-called *proximal operators* (we use ξ exclusively to denote the generic argument variable):

$$\mathbf{prox}_{f,\sigma}(\xi) := \arg\min_{\mathbf{x}} \left(f(\mathbf{x}) + \frac{\sigma}{2}\|\mathbf{x}-\xi\|^2 \right). \tag{5}$$

One such algorithm is the PD method [PCBC09], which solves a slightly more general problem:

$$\underset{\mathbf{x}}{\text{minimize}} \quad f(K\mathbf{x}) + g(\mathbf{x}) \tag{6}$$

for some linear operator K. PD solves the problem iteratively by providing a series of variable updates that terminate when $\mathbf{z}$ converges to the solution. The combination of $\mathbf{x}, \mathbf{z}$ and $\mathbf{y}$ ensures that $\mathbf{z}$ converges to the optimal value of Eq. 6. The updates are given by

$$\mathbf{x}^{k+1} := \mathbf{prox}_{f^*,1/\sigma}(\mathbf{x}^k + \sigma K\mathbf{y}^k) \tag{7}$$

$$\mathbf{z}^{k+1} := \mathbf{prox}_{g,1/\tau}(\mathbf{z}^k - \tau K^*\mathbf{x}^{k+1}) \tag{8}$$

$$\mathbf{y}^{k+1} := \mathbf{z}^{k+1} + \theta(\mathbf{z}^{k+1} - \mathbf{z}^k), \tag{9}$$

where $\{\sigma, \tau, \theta\}$ are parameters that affect convergence, f^* and K^* are the convex conjugates of f and K, respectively. For our problem, K is simply the identity, which leads to $K^* = K^T = I$. Note that if additionally $\sigma, \tau, \theta = 1$, the iterative update scheme reduces to ADMM. A more appropriate choice ($\sigma, \tau, \theta \neq 1$) leads to optimal control over convergence. As for f^*, it is not necessary to compute it directly. The proximal operator can be transformed using Moreau's identity:

$$\mathbf{prox}_{f^*,1/\sigma}(\xi) = \xi - \sigma\,\mathbf{prox}_{f,\sigma}(\xi/\sigma). \tag{10}$$

The variable updates are thus reduced to

$$\mathbf{x}^{k+1} := \mathbf{x}^k + \sigma\mathbf{y}^k - \sigma\,\mathbf{prox}_{f,\sigma}(\tfrac{1}{\sigma}\mathbf{x}^k + \mathbf{y}^k) \tag{11}$$

$$\mathbf{z}^{k+1} := \mathbf{prox}_{g,1/\tau}(\mathbf{z}^k - \tau\mathbf{x}^{k+1}) \tag{12}$$

$$\mathbf{y}^{k+1} := \mathbf{z}^{k+1} + \theta(\mathbf{z}^{k+1} - \mathbf{z}^k). \tag{13}$$

A more in-depth discussion of PD can be found in [CP11].

The advantage of using proximal methods is that the optimization can be performed separately for the two objective functions, allowing difficult optimizations to be split into more manageable components. Also, depending on the form of f and g, many special cases can be significantly simplified [BPC*11] by exploiting their mathematical structures.

To the best of our knowledge, PD has not yet been applied to fluid problems, despite its provably optimal convergence properties for the class of problems of Eq. (6). A pseudocode implementation of our PD-based optimization method is given in Algorithm (1) for one time step. Comparing to previous methods, the improved

convergence is achieved with only a minimal increase in computational cost. A few more vector additions and multiplications are necessary, which typically are negligible compared to the cost of the proximal operators. We demonstrate the convergence of our method and compare it to other methods in Sections 4.1 and 5.1. For reference, we briefly review IOP and ADMM in Appendix B.

Convex Optimization of Fluids In fluid simulation, we often encounter problems of the form

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in C_{\text{DIV}}, \end{aligned} \tag{14}$$

where $\mathbf{x}$ is the velocity field we seek, and C_{DIV} is the space of divergence-free velocity fields. f must be a convex function. Practically, this can be either a quantity we are trying to minimize, or a hard constraint that must be satisfied (in which case, f would be an indicator function).

The second constraint requires $\mathbf{x}$ to be divergence-free. Removing the divergent part of the flow can also be viewed as an orthogonal projection [Cho68, PB13]. Gregson et al. [GITH14] made the key observation that the proximal operator for C_{DIV} can be easily computed via a pressure projection. In other words, we have

$$\mathbf{prox}_{g,1/\tau}(\xi) = \Pi_{\text{DIV}}(\xi), \tag{15}$$

where Π_{DIV} denotes a projection onto C_{DIV} with a commonly used Poisson solver. Hence, formulating a fluid problem this way allows the optimization algorithm to be easily integrated into a common fluid solver—we simply replace the call for a pressure projection subroutine with a call to the PD optimization step outlined in Algorithm (1). We check for convergence using a threshold parameter ε, and stop the algorithm once the per-iteration change of $\mathbf{z}$ falls below this threshold (Algorithm (1), line 12). Note that we define the pressure projection to include only the calculation of the pressure values, and a subtraction of the pressure gradient from $\mathbf{x}$, excluding any optional modifications of the velocity field.

The pressure projection implemented by a CG solver is usually the most expensive part in regular fluid simulation, and its effect is magnified in our algorithm due to its iterative nature. However, this iterative set-up gives us an opportunity to apply an adaptive scheme. Let ε_{CG} be the accuracy of the CG solver. We choose its value starting with a high threshold (e.g., $\varepsilon_{\text{CG}} = 10^{-2}$) and then adaptively decrease it over the PD iterations. We decrease ε_{CG} as soon as the per-iteration change of $\mathbf{z}$ is close to ε; until ε_{CG} reaches the desired final accuracy (e.g., $\varepsilon_{\text{CG}} = 10^{-5}$). Using this adaptive CG scheme greatly accelerates the performance by cutting down on CG iterations in the beginning of the optimization when the divergence-free constraint does not need to be strictly enforced.

Algorithm 1 Our PD-based method for fluid simulation

1: **procedure** PD($\mathbf{u}_c$, $\mathbf{u}_t$, τ, σ, θ)
2: **while** $k < maxIters$ **do**
3: // $\mathbf{x}$-update
4: $\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \sigma\mathbf{y}^k - \sigma\text{proxF}(\sigma, \frac{1}{\sigma}\mathbf{x}^k + \mathbf{y}^k)$
5: // $\mathbf{z}$-update (using adaptive CG accuracy)
6: $\mathbf{z}^{k+1} \leftarrow \Pi_{\text{DIV}}(\mathbf{z}^k - \tau\mathbf{x}^{k+1})$
7: // $\mathbf{y}$-update
8: $\mathbf{y}^{k+1} \leftarrow \mathbf{z}^{k+1} + \theta(\mathbf{z}^{k+1} - \mathbf{z}^k)$
9: // check stopping criterion
10: $\mathbf{r}^{k+1} \leftarrow \mathbf{z}^{k+1} - \mathbf{z}^k$
11: $\varepsilon \leftarrow \sqrt{n_{\text{dim}}}\varepsilon_{\text{abs}} + \varepsilon_{\text{rel}} \left\|\mathbf{z}^{k+1}\right\|$
12: **if** $\left(\left\|\mathbf{r}^{k+1}\right\| \leq \varepsilon\right)$ **then break**
13: **return z**

Algorithm (1) summarizes the general framework of our method as it applies to fluid simulation. Specific applications call for different definitions of f, which in turn affects how the $\mathbf{x}$-update is computed. In the following sections, we discuss how to apply this method to the fluid guiding and the separating BC problem. For each application, we define the appropriate f and discuss how to compute its proximal operator.

4. Fluid Guiding

In fluid guiding, the goal is to minimize the change applied to the current velocity field such that the resulting velocity field follows the large-scale motions of a given target velocity. The objective function f is given by

$$f(\mathbf{x}) = \|G(\mathbf{x} - \mathbf{u}_t)\|^2 + \|W(\mathbf{x} - \mathbf{u}_c)\|^2, \tag{16}$$

where $\mathbf{u}_c$ is the current velocity field (after advection and before pressure projection), $\mathbf{u}_t$ is the target velocity field, $\mathbf{x}$ is the guided velocity field, W is the guiding weights matrix, and G is the Gaussian blur matrix. The first term of this objective function is minimal for a solution that matches the target velocity when blurred by G, while the second term penalizes solutions far away from the current flow field.

In order to keep the application general, both matrices in our objective function are spatially varying (Nielsen and Christensen [NC10] also performed fluid guiding with spatially varying guiding weights, but not with spatially varying blur). $W = W(\mathbf{x})$ is a diagonal matrix containing spatially varying weights that control the guiding strength (larger entries denote weaker guiding), and $G = G(\beta, \mathbf{x})$ is the Gaussian blur matrix that applies a blur of radius β only to fluid cells so that we can handle boundaries and obstacles in our domain.

Our objective function in Eq. (16) is quadratic. That is, it can be expressed as

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x} + c, \tag{17}$$

where

$$A = 2(G^T G + W^2) \tag{18}$$

$$\mathbf{b} = -2(G^T G\mathbf{u}_t + W^2\mathbf{u}_c) \tag{19}$$

$$c = \mathbf{u}_t^T G^T G\mathbf{u}_t + \mathbf{u}_c^T W^2\mathbf{u}_c. \tag{20}$$

Note that $W^T W = W^2$ since W is symmetric.

We can combine $f'(\mathbf{x}) = \mathbf{0}$ (three equations per cell in 3D) and the divergence-free constraint (one equation per cell) into a linear system $L\mathbf{x} = \mathbf{d}$. But since this system is overconstrained, we solve it in the least-squares sense by considering

$$L^T L\mathbf{x} = L^T\mathbf{d}, \tag{21}$$

where $L = \begin{pmatrix} A \\ \nabla \cdot \end{pmatrix}$ and $\mathbf{d} = \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix}$. This quadratic system can be solved using an iterative solver such as the CG solver. However, this approach is infeasible since the number of elements in the matrix grows by $\mathcal{O}(N)$, where N is the volume of the simulation domain. Thus PD is still a good choice even for seemingly simple quadratic energies. We will see later how this direct CG solver compares to our algorithm in terms of performance.

Instead, we make use of the quadratic property of f to simplify its proximal operator to

$$\mathbf{prox}_{f,\sigma}(\xi) = (A + \sigma I)^{-1}(\sigma\xi - \mathbf{b}). \tag{22}$$

Factoring out $\mathbf{u}_c$, we obtain

$$\mathbf{prox}_{f,\sigma}(\xi) = \mathbf{u}_c + M^{-1}(\sigma\xi + \mathbf{q}), \tag{23}$$

where

$$M = 2G^T G + 2W^2 + \sigma I \tag{24}$$

$$\mathbf{q} = 2G^T G(\mathbf{u}_t - \mathbf{u}_c) - \sigma\mathbf{u}_c. \tag{25}$$

Computing $M^{-1}(\sigma\xi + \mathbf{q})$ for every iteration is slow. Instead, we precompute $\mathbf{q}$ and M^{-1} since they do not rely on previous iterations. In fact, using the Sherman-Morrison-Woodbury formula, M^{-1} can be approximated as

$$M^{-1} \approx (2W^2 + \sigma I)^{-1} - 2(2W^2 + \sigma I)^{-1} G^T G (2W^2 + \sigma I)^{-1}. \tag{26}$$

See Appendix C for derivation details.

Lastly, spatially invariant Gaussian blurs are symmetric, so $G^T G = G^2$. They are also separable, and can be applied independently in each dimension. This combined with the matrix inversion approximation greatly speeds up the **x**-update. For Gaussian blurs with spatial variation, we still use a symmetric and hence separable Gaussian blur (with different blur radii) for each cell, but G itself is no longer symmetric. However, if the spatial variation is limited (e.g. different blur radii on the left and right sides of the simulation domain), we approximate G^2 by $G^T G$, since $G^T G \approx G^2$ for regions of constant blur. We found our approximation to work well in practice, and we will show examples using this approximation later. Our PD guiding scheme is independent of the particular choice and implementation of the blur kernel. Given a fast implementation for calculating G^T, it would be straight-forward to extend our method with a full evaluation of $G^T G$.

Additionally, our approach for applying the blur kernel easily allows for interior boundaries by using a blur radius of zero for obstacle cells. This is a key difference from guiding scheme based on the fast Fourier transform [GITH14], which typically require periodic domains without internal boundaries.

Algorithm (2) summarizes how $\mathbf{prox}_{f,\sigma}(\xi)$ is approximated. Note that the two extra parameters W and β in Algorithm (2) as compared to Algorithm (1) are specific to the guiding application.

Evaluation In this section, we will first demonstrate the efficacy of our method using 2D examples, followed by more impressive 3D results. To simplify the notation, when the guiding weight is spatially invariant (say, c everywhere), we will write $W = c$. And when the guiding weight is spatially varying (say, c_1 and c_2 on the left and right side on the simulation domain, respectively), we will write $W = (c_1, c_2)$. Similarly, spatially varying blur radii would be written as $\beta = (r_1, r_2)$.

Algorithm 2 Approximation for $\mathbf{prox}_{f,\sigma}(\xi)$ in fluid guiding

```
procedure PROXF(σ, ξ, W, β)
    q = 2G^T G(u_t − u_c) − σu_c // can be precomputed
    γ = (2W^2 + σI)^{-1} // inverse of diagonal matrix
    return u_c + γ(σξ + q) − 2G^T Gγ^2(σξ + q)
```

We first compare our method to two naïve guiding methods, with a counterclockwise circular velocity field as the target $\mathbf{u}_t$, as shown in Figure 2a. Linear velocity blend computes the new velocity as a linear combination of the current velocity and the target velocity: $\mathbf{u}_{\text{new}} = r\mathbf{u}_c + (1-r)\mathbf{u}_t$, where $0 \le r \le 1$. Detail-preserving guiding [TKRP06] subtracts the large-scale (i.e., blurred) motions from the current velocity before adding the target velocity: $\mathbf{u}_{\text{new}} = \mathbf{u}_c - G\mathbf{u}_c + \mathbf{u}_t$. The idea is to create a new velocity with large-scale motions from $\mathbf{u}_t$ and small-scale details from $\mathbf{u}_c$. Both methods are followed by a pressure solve to ensure zero divergence.

With linear velocity blend, guiding strength is increased with smaller r. The method's main weakness is that small fluid details are smoothed out with strong guiding, and more details come at the expense of reduced trajectory control (see Figure 2b).

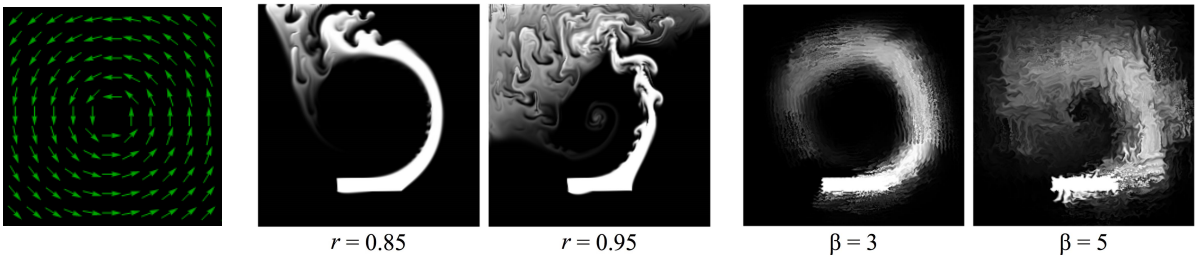
Detail-preserving blend allows guiding of large-scale motions towards the target velocity while preserving details. However, the details are difficult to control and have an unnatural frosted glass appearance (see Figure 2c).

In contrast, our method has much more flexible motion control when applied to fluid guiding, with W affecting the large-scale guiding strength and the blur radius β controlling the small-scale details. Figure 3 shows how the parameters affect guiding for a 2D smoke simulation with a circular target velocity. Notice that larger W values allow for more freedom to deviate from the target, while larger β values lead to the formation of larger vortices.

We also compare our algorithm to wavelet turbulence [KTJG08], as shown in Figure 4. Although wavelet turbulence is fast—since it is purely a postprocessing technique—the resulting vortices do not couple as tightly and realistically as with our method.

Performance Next, we examine our method in relation to other proximal methods, namely, ADMM and IOP (see Appendix B for details). ADMM is fairly straightforward to apply due to its similarity to PD. IOP can be interpreted as a fluid-specific version of the Projection onto Convex Sets (PoCS) algorithm [BB96], which solves convex feasibility problems (locating an intersection point of convex sets) rather than the more general problem solved by ADMM and PD. Applying IOP naïvely alternates between the minimizer of f and its closest divergence-free neighbor but does not yield the divergence-free minimizer of f, as illustrated by the failure case in Figure 5. In practice, we discovered instances for which IOP found plausible approximate solutions in a short time, but its reliability is limited for general guiding applications. As such, we focus on only comparing our method to ADMM. All performance measurements were done on PCs with Intel Xeon E5-1650 CPUs.

For fairness of comparison, we experimented with several test



(a) Target velocity (b) Linear velocity blend (c) Detail-preserving blend

Figure 2: Naïve guided simulations with circular targets.

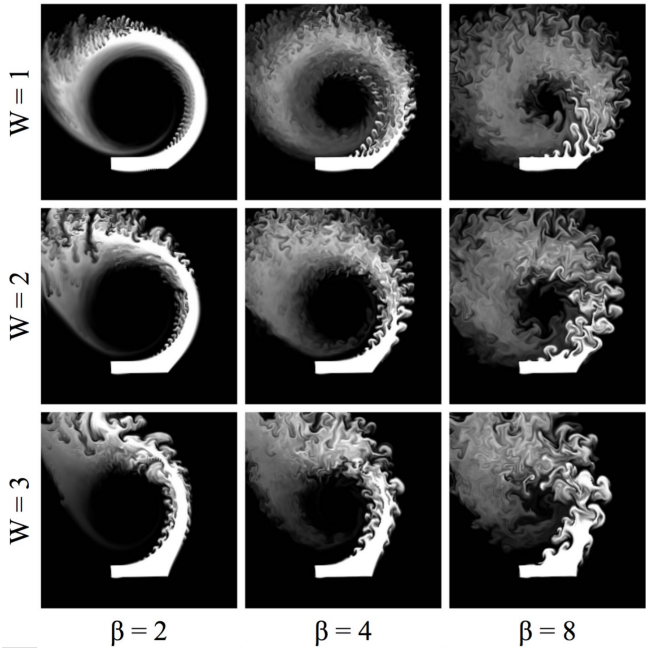


Figure 3: Guided simulations with circular targets, using our method with varying W and β for multi-scale motion control.

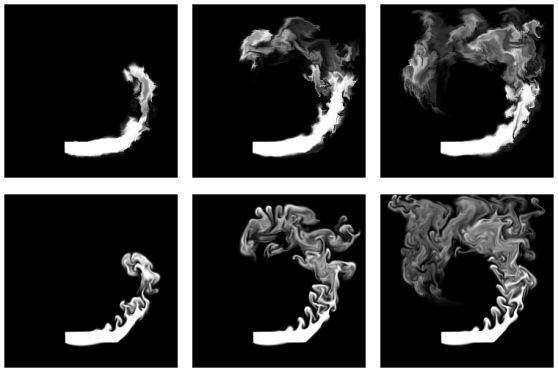


Figure 4: Upsampling from 64^2 to 256^2 using wavelet turbulence (top) and using our method (bottom). Our method yields significantly more coherent vortices.

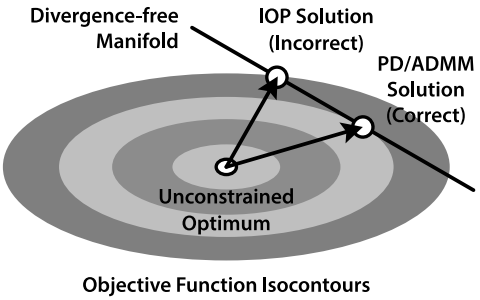
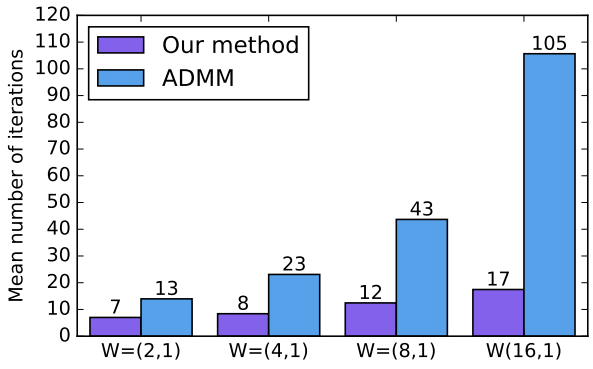


Figure 5: A divergent target velocity leads to IOP failure, while ADMM and PD converge successfully. In this case, even though the IOP solution is divergence-free, it lands on an isocontour farther from the center than the PD/ADMM solution, which is the true minimum.

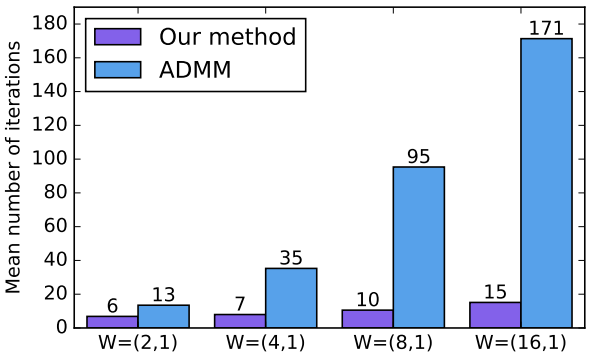
scenes to deduce as-optimal-as-possible parameter sets for both ADMM and our method. These parameters, on average, produce the fastest convergence rates under various guiding weights and blur radii. The optimal parameters are correlated with the guiding weight; we analyzed their relationship and define the parameters in terms of the average guiding weight, $\bar{W}$, which is simply the mean of the diagonal terms of W. For our method, we chose $(\tau, \sigma, \theta) = (0.58/\bar{W}, 2.44/\tau, 0.3)$, and for ADMM, $\rho = 1.4\bar{W}^2$. Before comparing their performance, it is important to note that we apply one of our contributions crucial for performance (the matrix inverse approximation and the separable Gaussians) to both methods.

We compare the performance of ADMM and our method for 2D and 3D problems. The 2D problem is a 256^2 simulation guided by a circular target velocity, similar to the one shown in Figure 3, but with spatially varying weights and $\beta = 1$. Specifically, the guiding weight is fixed at 1 for the right side of the domain, while the left side takes values from $\{2, 4, 8, 16\}$. Figure 6a compares, for the two methods, the mean number of iterations required to reach convergence. When the spatial variation is low, both methods perform decently. However, as the spatial variation increases, ADMM takes much longer to converge.

The 3D problem is a 120^3 simulation with an obstacle, similar to the one in Figure 13, but with the same W and β parameters as the 2D problem. The performance results are shown in Figure 6b.



(a) 2D circular example



(b) 3D obstacle example

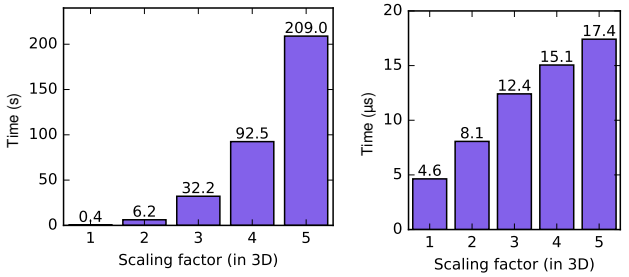
Figure 6: Mean number of iterations per time step for guiding.

		Guiding weight W			
		(2,1)	**(4,1)**	**(8,1)**	**(16,1)**
2D	**Our method**	0.14	0.17	0.26	0.38
	ADMM	0.26	0.41	0.76	1.89
3D	**Our method**	16.6	19.5	26.3	38.4
	ADMM	30.4	67.8	176.9	325.6

Table 1: Mean run time (in seconds) per time step for guiding.

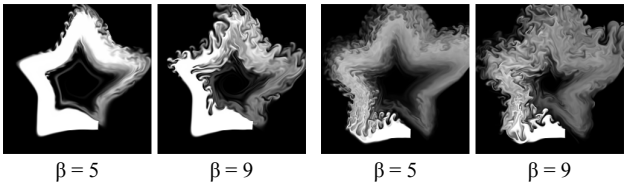
Again, our method outperforms ADMM. In fact, the difference in convergence rates is even more apparent for the 3D example for large spatial variations. The run times for both ADMM and our method very closely correlate with the mean iteration counts. For brevity, the run times are given in Table 1.

In addition to the comparison with ADMM, we also analyze how our method scales as the resolution increases. In Figure 7a, we run the guided 3D tornado simulation shown in Figure 1 at five different resolutions and compare the mean run time per time step. The scaling factor is with respect to the lowest resolution $40^2 \times 60$; for instance, a scaling factor of 3 corresponds to the resolution $120^2 \times 180$. Notice that even at the largest resolution $200^2 \times 300$, each time step takes less than four minutes to complete. However, since the run times appear to scale exponentially with respect to resolution, we plot the mean run time per grid cell (see Figure 7b) to show that the scaling is in fact linear.



(a) (b)

Figure 7: Performance of our method on the 3D tornado example at various resolutions, including (a) the mean run time per time step, and (b) the mean run time per time step per grid cell.



(a) with up-res (b) without up-res

Figure 8: Simulations guided by a star-shaped target.

Previously, we mentioned that applying a CG solver also works for the guiding problem, although its poor performance renders it infeasible. To test this, we ran a small 128^2 example with a circular target velocity field and found the direct CG solver (see Eq. (4)) approach to be 4000 times slower than our method.

The performance tests above are all done with our generic guiding method, which handles spatially varying weights and blurs. In the special case with spatially invariant weights, we can apply a non-iterative method to greatly speed up the simulation. The method involves defining $f(\mathbf{x})$ in terms of the non-divergent components of the input velocities and then finding its minimizer. Since differentiation commutes with convolution, the result will be automatically divergence-free for spatially invariant weights. Although this non-iterative method offers potential speed-up, we focus on the general case of spatially varying operators that requires iterating.

4.1. Guiding Results

To realize interesting guided flows, we found an iterative *up-res* workflow that works well in practice. This is in line with previous work, where a low-resolution input is refined in subsequent stages to yield a final result [KTJG08, YCZ11, HMK11, RLL*13, HK13]. Examples of this process are shown in Figure 8a. We start with a 64^2 simulation guided by a synthetic star-shaped velocity field with $W = 1$ and $\beta = 1$. Then we run 256^2 high-resolution simulations with various β values. If the same guiding is done without up-res, that is, if we run the 256^2 guided simulation directly as shown in Figure 8b, then we lose the ability to guide the fluid to desirable shapes at multiple levels.

We further demonstrate this up-res process for 3D simulations.

Figure 9 shows a simple plume example. We first run a low-resolution ($50^2 \times 100$) simulation to capture the velocities. Then a $200^2 \times 400$ resimulation is performed, guided by the upsampled velocity field. The blur radii can be kept spatially invariant or spatially varying for different effects. In particularly, increasing the blur radius introduces more small-scale details. In our spatially varying example, we use a blur kernel with a sharp transition between two blur radii to contrast between the two guided regions. If desired, this obvious seam can be softened via a more gradual transition between the two blur radii. Note that even though the simulation uses the approximation $G^T G = G^2$, it works quite well in practice.

Our second example in Figure 10 shows 3D simulations following star-shaped input velocities. We use a 50^3 target velocity as guide for a 250^3 final resolution, with $\beta = 2$. Here we make use of spatially varying weights, with $W = 1$ on the left side of the domain, and $W \in \{1, 5, 7\}$ on the right. The precomputation time is negligible compared to the overall run time: around 1.5 seconds per time step for a 250^3 simulation.

Next, we show a tornado simulation. We first run a low-resolution ($40^2 \times 60$) simulation using a cylindrical target velocity with a small upward component. Once the general shape is fixed, a $200^2 \times 300$ resimulation is performed, guided by the upsampled velocity field. The results in Figure 11 show the effectiveness of varying β to achieve different levels of turbulence.

Finally, Figure 12 demonstrates an example with obstacle, for which we upsampled from 40^3 to 200^3 with $W = 1$ and $\beta = 5$. This high-resolution version captures the input motion, but develops many interesting small-scale details. Figure 13 shows a similar simulation with $\beta = 1$ and spatially varying weights, where the left side has $W = 1$ while the right side has $W = 100$. The result has little detail on the left side due to strong guiding from the low-resolution simulation, whereas the right side behaves like a regular smoke simulation due to weak guiding. This guided simulation with an obstacle is a case that cannot be handled by previous Fourier-domain guiding schemes [GITH14] due to the interior boundaries. Arbitrary boundaries are easily handled by our separable approximation to the fluid guiding proximal operator while remaining highly efficient and easily parallelizable.

5. Separating Solid-Wall BCs

Another application of our PD-based method is the realization of separating solid-wall BCs. As a motivational example, consider Figure 14, which shows a 2D breaking dam simulation. When generated by a common CG pressure solver (top row), the fluid exhibits the undesirable behavior of sticking to the ceiling and getting stuck in corners. In contrast, our method (bottom row) features a clean separation of the fluid from all solids. In many visual effect settings that aim for large-scale fluid flow, the separation is preferable due to its improved realism.

We achieve this behavior by implementing separating solid-wall BCs in the form of an inequality constraint $\mathbf{u} \cdot \hat{n} \geq 0$ (with $\mathbf{u}$ and $\hat{n}$ denoting velocity and obstacle normal) without transforming the linear system of equations for the pressure solve into a more complicated problem. As outlined in Section 3, we split the problem into two simpler objective functions, with f controlling the BCs and g enforcing zero divergence (see Eq. (15)). With this setup, a regular CG solver is employed to compute g, while f is handled by an efficient projection and classification scheme, as described next.

Velocity BCs The proximal operator $\mathbf{prox}_{f,\sigma}(\xi)$ ensures that velocities at obstacle surfaces never point into the solid (i.e., negative normal velocity components at fluid-solid faces). Separating velocities (i.e., positive normal components), on the other hand, are allowed and thus left alone. We assume an obstacle is static and has a velocity of zero at its surface. BCs that form a linear constraint on the velocity field, like requiring velocity components to be zero, can be expressed as orthogonal projections [MCPN08]. Therefore, $\mathbf{prox}_{f,\sigma}(\xi)$ reduces to a projection onto the space of velocities without flows into obstacles. In its simplest form, this proximal operator sets normal velocity components to zero wherever $\mathbf{u} \cdot \hat{n} < 0$. The pressure solver, on the other hand, is unaware of any obstacle boundaries and uses free surface Dirichlet BCs at liquid surfaces. Later, we present an accelerated version that makes the pressure solver partially aware of obstacles. For now, all obstacles are fully handled by f.

Although this simple velocity projection works for exact solutions, it breaks down when solving up to finite accuracy. The pressure solver can accumulate small positive velocities during iterations of our optimization procedure, leading to small separating motions where the liquid should only be standing still or moving tangentially. This can happen for hydrostatic cases, for instance, where the final velocity at the wall should be exactly zero. We address this problem by introducing a classification step with hysteresis.

We assume by default that our cells are separating boundary cells with a Dirichlet $p = 0$ condition, and then create a list of cells that are explicitly not allowed to separate. For the classification, in each iteration of our algorithm, we accumulate all motions into a wall returned by the CG solver for a cell i and store it in m_i. This allows for a solid cell classification with temporal coherence. A non-separating cell only becomes separating if the magnitude of its velocity component away from the wall $\mathbf{u}_i \cdot \hat{n}_i$ exceeds the magnitude of m_i. Additionally, a change in the separation classification is only made if the absolute value of $\mathbf{u}_i \cdot \hat{n}_i$ is greater than the accuracy of the CG solver $\varepsilon_{\mathrm{CG}}$. Otherwise, the cell keeps its previous state. The last two steps effectively implement a hysteresis that prevents cells from changing status due to numerical errors, which is crucial for a reliable convergence of our method.

Figure 15 illustrates this process. It shows two possible fluid behaviors at the face of a solid: non-separating (left) and separating liquid (right). Both velocities fulfill $|\mathbf{u}_i \cdot \hat{n}_i| > \varepsilon_{\mathrm{CG}}$ (black bars), and can thus potentially change state. On the left of Figure 15, the cell face has a large accumulated value m_i (red arrow). Assuming that the face velocity $\mathbf{u}_i \cdot \hat{n}_i$ (green arrow) exceeds the threshold $\varepsilon_{\mathrm{CG}}$, the cell is classified as non-separating, and $\mathbf{u}_i \cdot \hat{n}_i$ is later on set to zero despite its positive value. In contrast, a positive normal velocity component greater than $|m_i|$ indicates that fluid is currently separating from the solid cell. The liquid boundary is treated as a free surface in this case, and its velocity is left unmodified.

Algorithm Summary Algorithm (3) shows pseudocode for the classification and the proximal operator for separating BCs. The

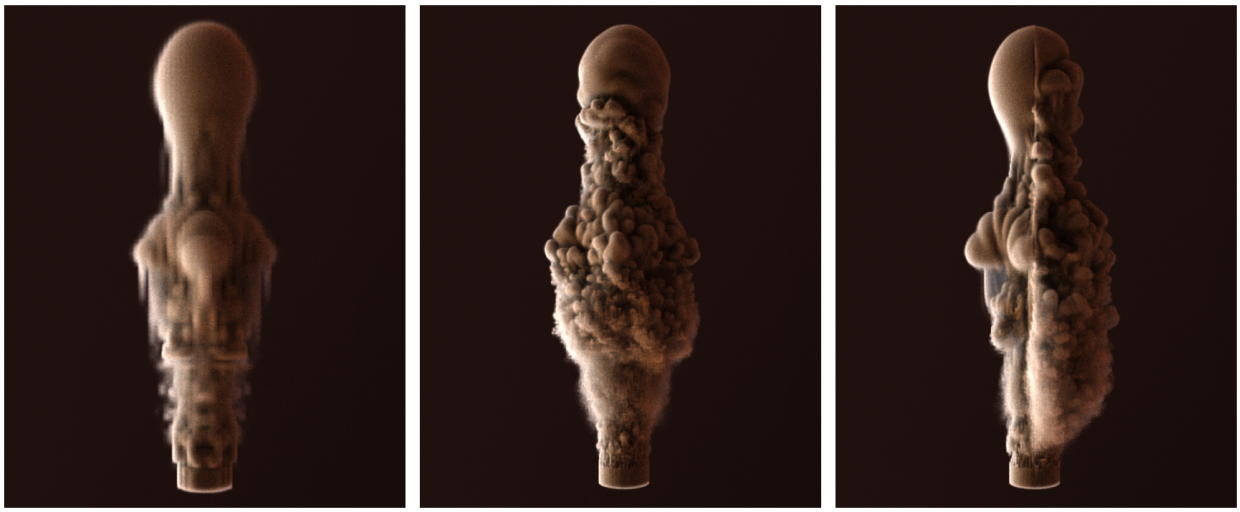
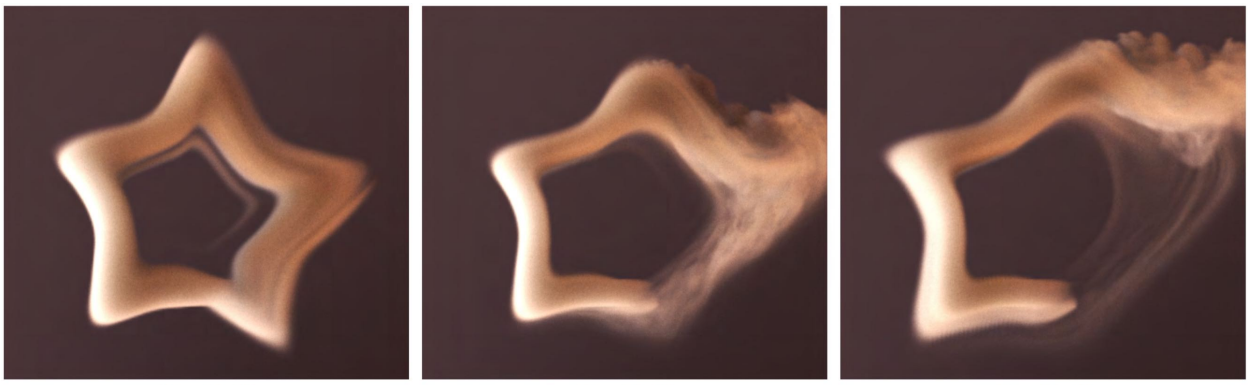
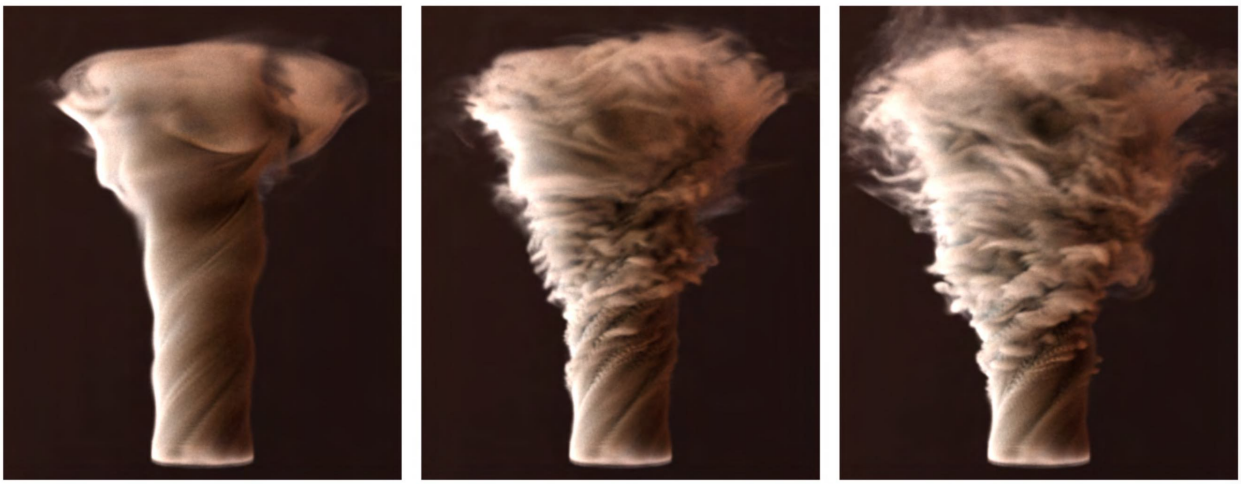


Figure 9: A $50^2 \times 100$ simple plume simulation (left) is upsampled to $200^2 \times 400$ with constant blur (middle, $\beta = 5$) and spatially varying blur (right, $\beta = (1, 10)$).



W = (1, 1) W = (1, 5) W = (1, 7)

Figure 10: A 250^3 star-shaped guided simulation upsampled from a 50^3 guided simulation with spatially varying weights.



β = 1 β = 5 β = 10

Figure 11: A $200^2 \times 300$ guided simulation upsampled from a $40^2 \times 60$ guided simulation with varying β.

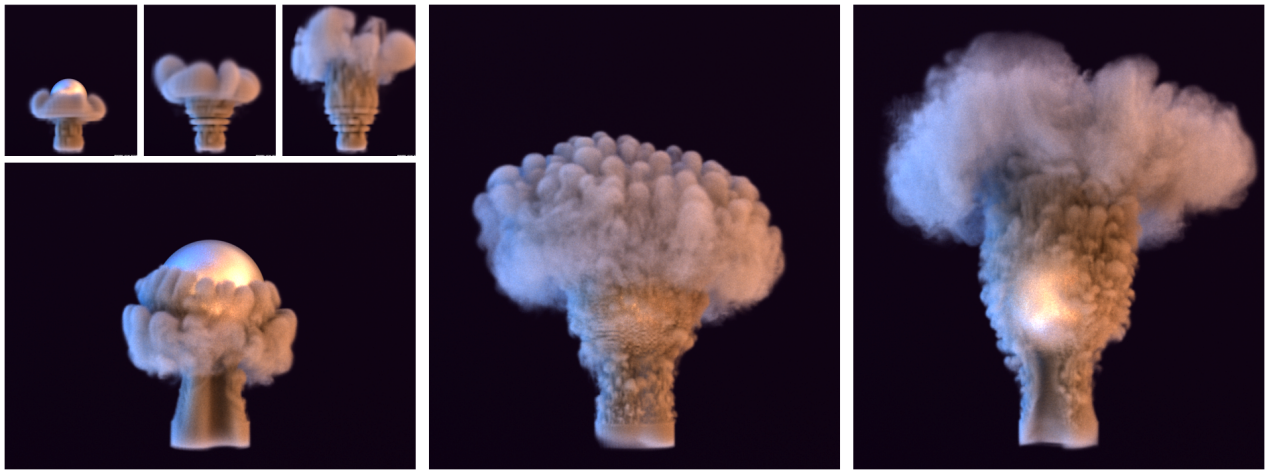


Figure 12: A 40^3 simulation with an obstacle (inset) is upsampled to 200^3 with $W = 1$ and $\beta = 5$.

$W = (1, 100), \beta = 1$

Figure 13: Guided obstacle scene with spatially varying weights.

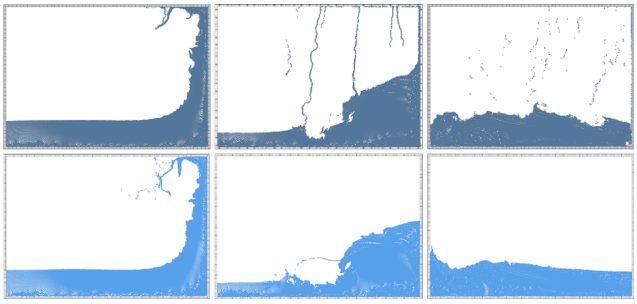


Figure 14: 200×140 Breaking Dam, Non-Separating (top) versus Separating BCs (bottom) at $t = 75, 112, 150$.

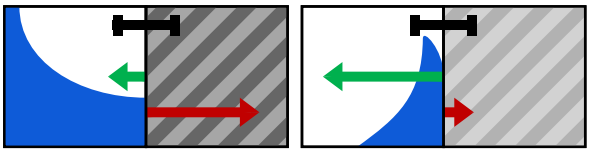


Figure 15: Classification of solid cells into non-separating (left) and separating (right). The green and red arrows denote $\mathbf{u}_i \cdot \hat{n}_i$ and m_i respectively, while the black bars indicate ε_{CG}.

classification proceeds as outlined above, updating the list of non-separating cells S_{nsep} based on the velocity field $\mathbf{u}$. The normal velocity components at non-separating cell faces are set to zero in the proximal operator $\mathbf{prox}_{f,\sigma}(\xi)$. The variable ξ is a generic argument variable being a combination of multiple velocity fields, including dual variables.

The PD algorithm solving for separating BCs follows the generic PD implementation outlined in Algorithm (1). We exchange the generic proximal operator by our BC specific projection from Algorithm (3). Additionally, we call the classification function with $\mathbf{z}$ after line 6 of Algorithm (1).

We adopt two smaller modifications from previous work that improve convergence. The first is a Krylov method from IOP [MCPN08], which we identified to work nicely within our BC solver (the full algorithm can be found in Appendix B). Additionally, we use the adaptive parameter scheme [CP11] for dynamic values of τ, σ and θ. The parameter updates are the following after choosing a suitable value for γ, τ^0 and σ^0 (we use $\gamma = 200$, $\tau^0 = 150$, $\sigma^0 = 1/\tau^0$): $\theta^k \leftarrow 1/\sqrt{1 + 2\tau^{k-1}\gamma}$, $\tau^k \leftarrow \tau^{k-1}\theta^k$, and $\sigma^k \leftarrow \sigma^{k-1}/\theta^k$.

Algorithm 3 Classification and $\mathbf{prox}_{f,\sigma}$ for BCs

procedure CLASSIFY($\mathbf{u}$, m, ε_{CG}, S_{nsep})
  for all boundary faces i, with $|\mathbf{u}_i \cdot \hat{n}_i| \geq \varepsilon_{CG}$ **do**
    if $\mathbf{u}_i \cdot \hat{n}_i \leq 0$ **then**
      $S_{nsep} \leftarrow S_{nsep} \cup \{i\}$ // non-separating
      $m_i \leftarrow m_i + \mathbf{u}_i \cdot \hat{n}_i$
    else if ($|\mathbf{u}_i \cdot \hat{n}_i| \geq |m_i|$) **then**
      $S_{nsep} \leftarrow S_{nsep} \setminus \{i\}$ // make separating
      $m_i \leftarrow 0$
procedure PROXF(ξ, S_{nsep})
  // velocity projection for marked cells
  for all $i \in S_{nsep}$ **do** $\xi_i \leftarrow \xi_i - (\xi_i \cdot \hat{n})\hat{n}$
  return ξ

Both methods reduce the overall run time, but are specific extensions for proximal operators given by orthogonal projections. As such, we use the extensions for all BC problems, but not for our guided simulations.

Accelerated BC Solver So far, the solid-wall BC handling is fully done in f while g simply ensures incompressibility. This is in line with the standard splitting approach in other methods [MCPN08, Hen12], and we demonstrate in the next section that our optimization scheme very efficiently calculates the solution in this case.

However, in practice, the total number of iterations can be reduced significantly by letting the pressure solver take care of all BCs that it is capable of enforcing correctly: retaining the input normal velocity components at fluid-solid faces by enforcing a zero pressure gradient at walls. For this version of our solver, we update the BCs of the pressure solver in g in accordance with our classification: Neumann BCs for non-separating cells, and Dirichlet (free surface) BCs for separating ones. This effectively locks the classification after few iterations, leading to a stable solution within the next iteration. However, it prevents non-separating cells from changing their state back to separating ones. Thus, the accelerated BC solver does not yield the same result as the standard version, but achieves much higher performance while featuring only negligible differences. A summary of our accelerated solver with pseudo-code can be found in Appendix D.

5.1. Evaluation and Results

We now evaluate the performance of our BC solver, and demonstrate the importance of allowing liquid to separate from walls in high-resolution simulations.[†] To validate that our PD solver yields the correct results, we use our method to calculate regular non-separating boundaries for a 200×140 breaking dam setup, which can also be handled by a regular CG solver. In this case, we can achieve arbitrary accuracies depending on the choice of parameters for CG and our scheme. These experiments show that our method converges to the correct solution.

[†] Thresholds for our algorithm are set to $\varepsilon_{CG} = 10^{-5}$ (as liquids are more sensitive to mass loss) and $\varepsilon_{abs} = \varepsilon_{rel} = 10^{-3}$.

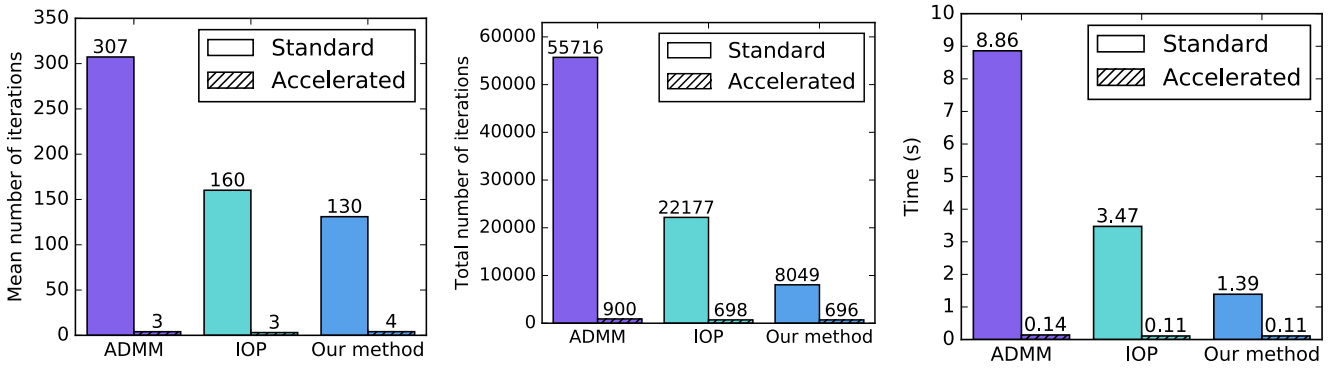
For separating boundaries, we can no longer compare to a regular CG solver. Instead, we evaluate the performance of our method against methods from previous convex optimization work: IOP and ADMM. IOP has been applied to enforce non-divergence and complex BCs simultaneously in [CMF12]. For the standard BC solver simulating a 2D breaking dam, our method converges six times faster than ADMM, and more than twice as fast as IOP, as shown in Figure 16c. Figure 16a and Figure 16b show the mean number of IOP/ADMM/PD and CG iterations. Our method generally requires a lower number of PD iterations compared to ADMM and IOP. Due to our adaptive CG accuracy scheme, the number of PD iterations is increased while the total amount of CG iterations is decreased. Both numbers influence the performance, but the total number of CG iterations more strongly influences runtime as shown in Figure 16c. Similar behavior is observed for a 3D complex breaking dam simulation (see Figure 18 for a visual example). The run time measurements are shown in Figure 17 where ADMM is omitted due to its impractical run time. The accelerated solver speeds up all methods significantly. In this case, our PD-based method performs on par with IOP, which we attribute to the low number of iterations required in this case; the higher-order convergence of the PD method does not develop its full potential in such a setting.

Figure 18 highlights that regular non-separating walls often yield undesirable results in practical settings. We simulate a complex 3D scene with liquid splashing onto multiple obstacles with a resolution of $256 \times 230 \times 256$. A regular solver leads to large amounts of liquid crawling along ceilings, and sticking to walls, while our separating boundaries give a much more believable large scale look, with liquid naturally separating from obstacle boundaries. For this setting, our accelerated BC solver requires $38.7s$ on average per frame. Comparing the overall time to generate this result (including surface generation) to a version with a regular pressure solver, our method increases run time by only 12%, with the added benefit of enabling separating BCs. This is a very practical result, considering that it was achieved based on a regular CG solver, without the need for specialized methods, such as Linear Complementarity Problems solvers [GB13]. Although solving LCPs has been studied in detail, these methods still fall into complexity classes that make large scale solves infeasible. An indication for this can be found in [GB13], where solving times of ca. 25s per LCP solve were given for a 100^3 example.

6. Discussion and Conclusion

We presented a general framework for incorporating the Primal-Dual method into fluid simulation, and demonstrated two applications: fluid guiding and separating BCs. We proposed a generic version of the Primal-Dual based optimization scheme with fast convergence for general proximal operator subproblems. In addition, we discussed several extensions that are particularly well-suited for optimizations with orthogonal projections as subproblems. Additionally, we demonstrated a novel formulation for the flow guiding problem, and an efficient approach for simulating liquids with separating BCs.

Limitations One limitation of our fluid guiding method is a lack of shape controls. Unlike smoke, liquids can require shape constraints to achieve a desired outline or shape. We have focused on



(a) IOP/ADMM/PD Iter

(b) Sum of CG Iter

(c) Runtime in seconds

Figure 16: 2D breaking dam, mean quantities per time step.

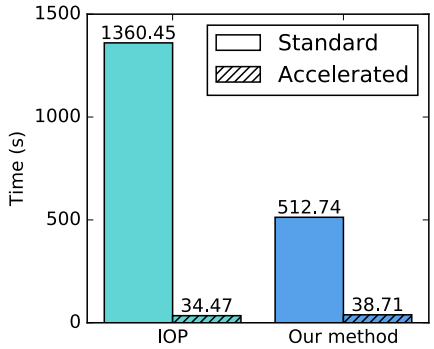


Figure 17: 3D complex breaking dam, mean run time per time step.

velocity guiding in this work, so controlling the shape of liquids will require extensions that take the position of the liquid's surface into account. Such constraints should integrate well into our overall pipeline, and we plan to investigate this topic as future work. Another area of improvement is bridging the gap between our optimization framework and an artist's workflow. Although we gave examples of different target velocities used for guiding, there is still a lot more to explore in terms of how to achieve various artistic visual effects intuitively. A limitation of our accelerated separating BC solver is that it can lead to slight deviations from the accurate, standard solution. We have not encountered visual artifacts resulting from these inaccuracies, and we believe that the improved performance typically outweighs these slight deviations. The specialized multigrid solvers [CMF12] could possibly outperform our BC solver. We believe that the attractiveness of our BC solver stems from its modularity and fast convergence. Adding support for wall separating BCs given an existing pressure solver requires little code with our method.

Outlook Our method is a very generic approach applicable to a large range of problems in fluid simulation. The price of this generality is that it may not be as fast as specialized methods tailored to specific problems. However, the modularity of our approach makes it easy to incorporate into existing implementations. As such, it has the potential to add powerful functionality, such as high-level flow guiding, into existing solvers without the need for complicated extensions.

Furthermore, there is a large number of interesting avenues to be explored with high-level optimizations of fluids flows. For example, we are interested in exploring shape optimization to adapt the geometry of obstacles with respect to their flow properties, and partial resimulations of regions in a flow.

7. Acknowledgments

This work was funded by the ERC Starting Grant *realFlow* (StG-2015-637014) and the NSERC Postdoctoral Fellowships Program.

Appendix A: Notation

For reference we provide a quick description of our notation. More detailed descriptions can be found in the body of the paper.

- f: first objective function in PD (application-dependent)
- g: second objective function in PD for incompressibility
- ξ: general input velocity field for several algorithms, a combination of several velocity fields, possibly dual variables
- $\mathbf{x}$, $\mathbf{z}$, $\mathbf{y}$: variables with iterative updates in PD
- τ, σ, θ: PD parameters controlling convergence rate
- α, β, G, $\mathbf{q}$, M, $\mathbf{u}_t$, $\mathbf{u}_c$, L: variables specific to fluid guiding
- $\mathbf{u}$: fluid velocity for the BC problem
- $\hat{n}$: normal of a solid cell
- m: memory velocity field
- S_{nsep}: set of non-separating fluid-solid faces

Appendix B: ADMM and IOP

Iterated Orthogonal Projection (IOP) [MCPN08]—a method similar to von Neumann's alternating projections [BPC*11]—requires both subproblems to be expressed as orthogonal projections

$$\mathbf{x}^{k+1} = \Pi_f(\mathbf{z}^k) \tag{27}$$

$$\mathbf{z}^{k+1} = \Pi_g(\mathbf{x}^{k+1}). \tag{28}$$

The Krylov method can improve its convergence rate:

1: **procedure** KRYLOV($\mathbf{z}^k$, $\mathbf{z}^{k-1}$, k, ε^{k-1})
2: $\quad\varepsilon^k = \text{error}(\mathbf{z}^k)$
3: $\quad$**if** $k > 1$ **then**

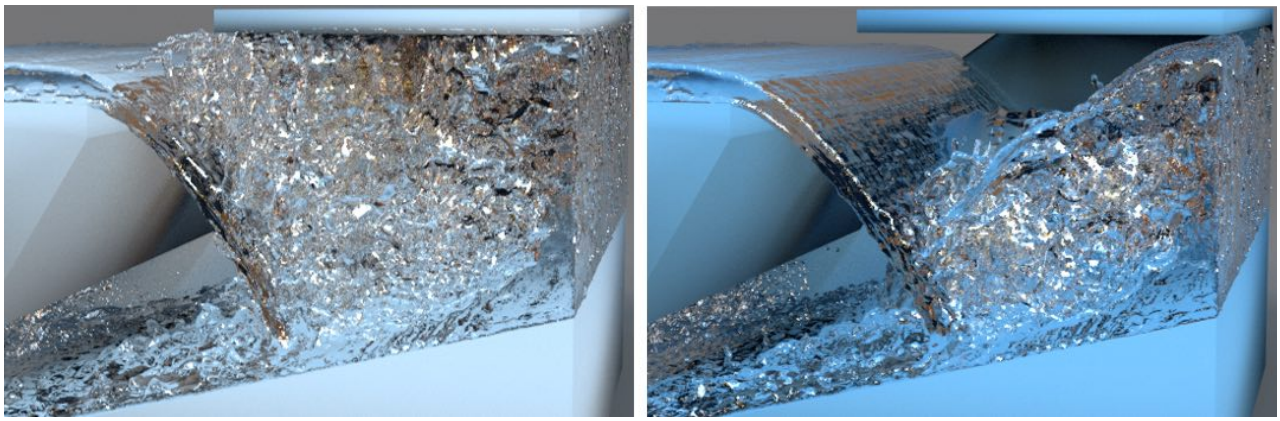


Figure 18: A $256 \times 230 \times 256$ scene with complex obstacles. Regular, non- BCs lead to the liquid sticking to walls (left), while our solver allows for naturally separating liquids (right).

$\mathbf{z}_{\varepsilon_{\text{dif}}} = \mathbf{z}^k - \mathbf{z}^{k-1}$ // correction vector
$\varepsilon_{\text{ratio}} = \varepsilon^k / \varepsilon^{k-1}$
$\mathbf{z}_{\varepsilon_{\text{tmp}}} = \mathbf{z}^k - \varepsilon_{\text{ratio}} \mathbf{z}_{\varepsilon_{\text{dif}}}$
$\varepsilon_{\varepsilon_{\text{tmp}}} = \text{error}(\mathbf{z}_{\varepsilon_{\text{tmp}}})$
if $\varepsilon^{\varepsilon_{\text{tmp}}} < \varepsilon^k$ **then** $\mathbf{z}^k = \mathbf{z}_{\varepsilon_{\text{tmp}}}$

The Alternating Direction Method of Multipliers (ADMM) [BPC*11, GOSB14] is a proximal method more general than IOP given by

$$\mathbf{x}^{k+1} := \mathbf{prox}_{f,\rho}(\mathbf{z}^k - \mathbf{y}^k) \tag{29}$$

$$\mathbf{z}^{k+1} := \mathbf{prox}_{g,\rho}(\mathbf{x}^{k+1} + \mathbf{y}^k) \tag{30}$$

$$\mathbf{y}^{k+1} := \mathbf{y}^k + \mathbf{x}^{k+1} - \mathbf{z}^{k+1}. \tag{31}$$

Instead of the three parameters $\{\tau, \sigma, \theta\}$ in PD that control convergence rate, ADMM only has one such parameter, ρ.

Appendix C: Inverse Matrix Approximation for Fluid Guiding

Here we present the details for deriving the inverse matrix approximation in Eq. (26).

We want to invert a matrix of the form $M = A + (2G^T G)$, where $A = 2W^2 + \sigma I$ contains the large diagonal terms and $2G^T G$ the small off-diagonal terms. By the Sherman-Morrison-Woodbury Formula,

$$M^{-1} = A^{-1} - 2A^{-1}G^T(I + 2GA^{-1}G^T)GA^{-1}. \tag{32}$$

Now since G and A^{-1} both contains small value entries, $2GA^{-1}G^T$ is approximately zero. Hence

$$M^{-1} \approx A^{-1} - 2A^{-1}G^T GA^{-1} \tag{33}$$

$$= (2W^2 + \sigma I)^{-1} - 2(2W^2 + \sigma I)^{-1} G^T G (2W^2 + \sigma I)^{-1}. \tag{34}$$

Note that the calculation of A^{-1} is trivial since $A = 2W^2 + \sigma I$ is diagonal.

Appendix D: Accelerated BC Solver

Below, we summarize our accelerated solver for separating solid-wall BCs. Instead of fully handling the solid-wall BCs in a proximal operator outside of the pressure projection, our accelerated solver employs the commonly used Neumann BCs during the pressure projection depending on the current classification of solid-wall cells. First, all solid wall cells are classified as separating (i.e. Dirichlet BCs for the pressure solver). Depending on the initial velocity field, solid wall cells with $\mathbf{u} \cdot \hat{n} < 0$ are classified as non-separating cells (Neumann BCs for the pressure solver). We then iteratively set the velocity BCs, apply a regular CG pressure solver and re-classify solid cells with the CLASSIFY procedure.

In our standard BC solver (Section 5), a memory field m is used to determine whether a cell is allowed to change its state to separating. In the accelerated case, m is not used, instead the pressure solver enforces solid wall BCs for non-separating cells. This prevents the normal velocity from becoming positive, and from changing its state back to separating. In the following procedure, m is replaced with the placeholder "$\cdot$".

procedure SOLVEPRESSUREWITHSEPARATINGBCS($\mathbf{u}$, ε)
  $S_{\text{nsep}} = 0$ // no cell is classified as non-separating yet
  CLASSIFY($\mathbf{u}$, $\cdot$, ε, S_{nsep})
  while S_{nsep} did not change in last call of CLASSIFY **do**
    PROXF$_{BC}$($\mathbf{u}$, S_{nsep})
    Π_{DIV}($\mathbf{u}$, ε)
    CLASSIFY($\mathbf{u}$, $\cdot$, ε, S_{nsep})

References

[BB96] BAUSCHKE H. H., BORWEIN J. M.: On projection algorithms for solving convex feasibility problems. *SIAM review 38*, 3 (1996), 367–426. 5

[BBB07] BATTY C., BERTAILS F., BRIDSON R.: A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. 26*, 3 (July 2007). 2, 3

[BHN07] BRIDSON R., HOURIHAM J., NORDENSTAM M.: Curl-noise for procedural fluid flow. *ACM Transactions on Graphics (TOG) 26*, 3 (2007), 46. 2

[BPC*11] BOYD S., PARIKH N., CHU E., PELEATO B., ECKSTEIN J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn. 3*, 1 (Jan. 2011), 1–122. 2, 3, 12, 13

[Bri08] BRIDSON R.: *Fluid Simulation for Computer Graphics*. AK Peters/CRC Press, 2008. 2

[Cho68] CHORIN A. J.: Numerical solution of the Navier-Stokes equations. *Mathematics of computation 22*, 104 (1968), 745–762. 4

[CMF12] CHENTANEZ N., MUELLER-FISCHER M.: A multigrid fluid pressure solver handling separating solid boundary conditions. *Visualization and Computer Graphics, IEEE Transactions on 18*, 8 (2012), 1191–1201. 3, 11, 12

[CP11] CHAMBOLLE A., POCK T.: A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision 40*, 1 (2011), 120–145. 3, 10

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *Proceedings of ACM SIGGRAPH* (2001), ACM, pp. 23–30. 2, 3

[FL04] FATTAL R., LISCHINSKI D.: Target-driven smoke animation. In *ACM Transactions on Graphics (TOG)* (2004), vol. 23, ACM, pp. 441–448. 3

[GB13] GERSZEWSKI D., BARGTEIL A. W.: Physics-based animation of large-scale splashing liquids. *ACM Trans. Graph. 32*, 6 (2013), 185. 3, 11

[GITH14] GREGSON J., IHRKE I., THUEREY N., HEIDRICH W.: From capture to simulation: connecting forward and inverse problems in fluids. *ACM Trans. Graph. 33*, 4 (2014), 1–11. 2, 4, 5, 8

[GOSB14] GOLDSTEIN T., O'DONOGHUE B., SETZER S., BARANIUK R.: Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences 7*, 3 (2014), 1588–1623. 13

[GP00] GILES M. B., PIERCE N. A.: An introduction to the adjoint approach to design. *Flow, turbulence and combustion 65*, 3-4 (2000), 393–415. 2

[HDN*16] HEIDE F., DIAMOND S., NIESSNER M., RAGAN-KELLEY J., HEIDRICH W., WETZSTEIN G.: ProxImaL: Efficient Image Optimization using Proximal Algorithms. *ACM Transactions on Graphics (TOG) 35*, 4 (2016). 2

[Hen12] HENDERSON R. D.: Scalable fluid simulation in linear time on shared memory multiprocessors. In *Proceedings of the Digital Production Symposium* (2012), DigiPro '12, ACM, pp. 43–52. 3, 11

[HK13] HUANG R., KEYSER J.: Automated sampling and control of gaseous simulations. *The Visual Computer 29*, 6 (2013), 751–760. 3, 7

[HMK11] HUANG R., MELEK Z., KEYSER J.: Preview-based sampling for controlling gaseous simulations. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2011), SCA '11, ACM, pp. 177–186. 3, 7

[HRH*13] HEIDE F., ROUF M., HULLIN M. B., LABITZKE B., HEIDRICH W., KOLB A.: High-quality computational imaging through simple lenses. *ACM Trans. Graph. 32*, 5 (Oct. 2013), 149:1–149:14. 2

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH* (1990), ACM Press, pp. 49–57. 2

[KTJG08] KIM T., THUEREY N., JAMES D., GROSS M.: Wavelet Turbulence for Fluid Simulation. *ACM Trans. Graph. 27*, 3 (2008), article 50. 2, 3, 5, 7

[MCPN08] MOLEMAKER J., COHEN J. M., PATEL S., NOH J.: Low viscosity flow simulations for animation. In *Proceedings of the Symposium on Computer Animation* (2008), Eurographics Association, pp. 9–18. 2, 8, 10, 11, 12

[MM13] MACKLIN M., MÜLLER M.: Position based fluids. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 104. 2

[MTPS04] MCNAMARA A., TREUILLE A., POPOVIĆ Z., STAM J.: Fluid Control Using the Adjoint Method. *ACM Trans. Graph. 23*, 3 (2004), 449–456. 2

[NB11] NIELSEN M. B., BRIDSON R.: Guide shapes for high resolution naturalistic liquid simulation. *ACM Transactions on Graphics (TOG) 30*, 4 (2011), 83. 2

[NC10] NIELSEN M. B., CHRISTENSEN B. B.: Improved variational guiding of smoke animations. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 705–712. 3, 4

[NCZ*09] NIELSEN M. B., CHRISTENSEN B. B., ZAFAR N. B., ROBLE D., MUSETH K.: Guiding of Smoke Animations through Variational Coupling of Simulations at Different Resolutions. In *Proceedings of the Symposium on Computer animation* (2009), pp. 217–226. 3

[NGCL09] NARAIN R., GOLAS A., CURTIS S., LIN M. C.: Aggregate dynamics for dense crowd simulation. In *ACM Transactions on Graphics (TOG)* (2009), vol. 28, ACM, p. 122. 3

[NGL10] NARAIN R., GOLAS A., LIN M. C.: Free-flowing granular materials with two-way solid coupling. *ACM Transactions on Graphics (TOG) 29*, 6 (2010), 173. 3

[NOB16] NARAIN R., OVERBY M., BROWN G. E.: ADMM $\supseteq$ projective dynamics: Fast simulation of general constitutive models. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2016), Eurographics Association, pp. 21–28. 2

[OV14] O'CONNOR D., VANDENBERGHE L.: Primal-dual decomposition by operator splitting and applications to image deblurring. *SIAM Journal on Imaging Sciences 7*, 3 (2014), 1724–1754. 2

[PB13] PARIKH N., BOYD S.: Proximal Algorithms. *Foundations and Trends in Optimization 1*, 3 (2013), 123–231. 4

[PCBC09] POCK T., CREMERS D., BISCHOF H., CHAMBOLLE A.: An algorithm for minimizing the mumford-shah functional. In *Computer Vision* (2009), IEEE, pp. 1133–1140. 2, 3

[PHT*13] PAN Z., HUANG J., TONG Y., ZHENG C., BAO H.: Interactive localized liquid motion editing. *ACM Trans. Graph. 32*, 6 (2013), 184:1–184:10. 2

[PTSG09] PFAFF T., THUEREY N., SELLE A., GROSS M.: Synthetic turbulence using artificial boundary layers. In *ACM Transactions on Graphics (TOG)* (2009), vol. 28, ACM, p. 121. 2

[RLL*13] REN B., LI C.-F., LIN M. C., KIM T., HU S.-M.: Flow field modulation. *IEEE Transactions on Visualization and Computer Graphics 19*, 10 (2013), 1708–1719. 3, 7

[Sta99] STAM J.: Stable Fluids. In *Proceedings ACM SIGGRAPH* (1999), pp. 121–128. 2

[SY05a] SHI L., YU Y.: Controllable smoke animation with guiding objects. *ACM Trans. Graph. (TOG) 24*, 1 (2005), 140–164. 3

[SY05b] SHI L., YU Y.: Taming Liquids for Rapidly Changing Targets. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation* (2005), pp. 229–236. 3

[TKRP06] THUEREY N., KEISER R., RUEDE U., PAULY M.: Detail-Preserving Fluid Control. In *Proceedings of the Symposium on Computer animation* (2006), pp. 7–12. 3, 5

[YCZ11] YUAN Z., CHEN F., ZHAO Y.: Pattern-guided smoke animation with lagrangian coherent structure. *ACM Trans. Graph. 30*, 6 (2011), 136. 2, 3, 7

[ZB05] ZHU Y., BRIDSON R.: Animating Sand as a Fluid. *ACM Trans. Graph. 24*, 3 (2005), 965–972. 2