Contents

Cell-Constrained Particles for Incompressible Fluids

ZOHAR LEVI

FLIP
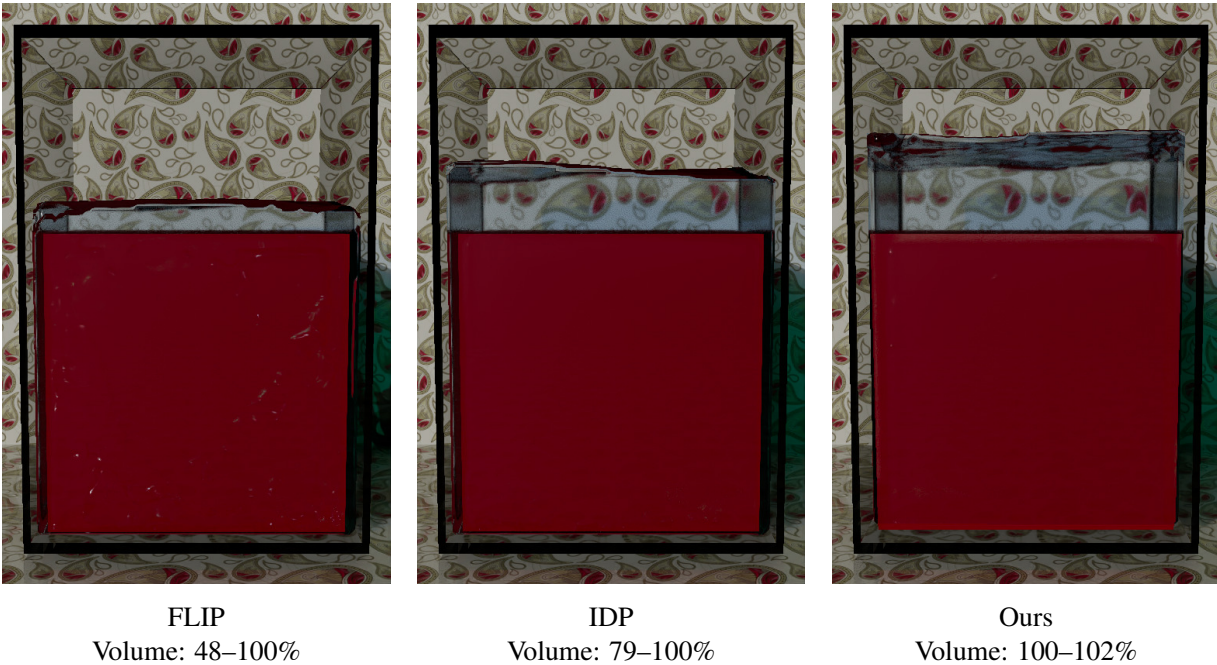Volume: 48–100%

IDP
Volume: 79–100%

Ours
Volume: 100–102%

Fig. 1. A large box is dropped into water. For FLIP [Zhu and Bridson 2005] and IDP [Kugelstadt et al. 2019], particles on the bottom of the tank do not manage to clear path in time, and they are trapped inside the solid box, leading to significant volume loss. The volume range over all time steps is indicated below each image.

Incompressibility is a fundamental condition in most fluid models. Accumulation of simulation errors violates it and causes volume loss. Past work suggested correction methods to battle it. These methods, however, are imperfect and in some cases inadequate. We present a method for fluid simulation that strictly enforces incompressibility based on a grid-related definition of discrete incompressibility.

We formulate a linear programming (LP) problem that bounds the number of particles that end up in each grid cell. A variant of the band method is offered for acceleration, which requires special constraints to ensure volume preservation. Further acceleration is achieved by simplifying the problem and adding a special band correction step that is formulated as a minimum-cost flow problem (MCFP). We also address coupling with solids in our framework and demonstrate advantages over prior work.

1 INTRODUCTION

Fluids behave in a rich and complex way. Simulating them faithfully has been an active research in computer graphics to improve the realism of an animation of a large variety of materials. Incompressibility is a fundamental condition in a fluid model, and it is expressed via a divergence-free constraint, which leads to volume conservation. The constraint restricts instantaneous movements of particles, and it has no long term view of a fluid during a simulation. No matter how accurate a simulation is, inaccuracies and numerical

errors accumulate overtime and become pronounced, and nothing accounts for changes in the fluid's volume.

To address that, correction methods have been offered. They range from improving particle spacing [Ando, Thurey, et al. 2012] to a stricter density correction via an additional solution of a Poisson equation [Kugelstadt et al. 2019], or treating particles as volume parcels with prescribed volume [Qu et al. 2022]. While some of the methods are quite effective, they are still not perfect, and in certain cases they are inadequate. We take a discrete approach to the problem, which restricts particles to grid cells, and show advantages over the state of the art.

We start by defining discrete incompressibility based on grid resolution (section 4). We offer a correction step to the PIC/FLIP framework [Bridson 2015] that preserves this discrete incompressibility in each time step. We limit movements of particles to specific locations in a local neighborhood of grid cells (section 5). Incompressibility is expressed through discrete constraints that bound the number of particles that end up in each cell. These are added as hard constraints to an integer linear programming (ILP) problem, which we show can be relaxed to a linear programming (LP) problem to improve running time. Nevertheless, the problem does not scale well, and running time can be an issue even for moderate size 3D grids. We offer a variant of the band method [Ferstl et al. 2016], tailored to our approach to preserve incompressibility (section 6).

Author's address: Zohar Levi.

This requires monitoring the amount of particles that go into and out of the band. For that, we formulate an additional special band constraint to restrict particle movements near the band interface. Two faster variants are offered. Both first solve the LP as is (without the additional band constraint), which results in an easier problem that is faster to solve. This is followed by a second specialized step to correct the band interface and deeper. One variant solve the LP again with a shorter one-way band constraint (section 6.2). The other variant formulates a minimum-cost flow problem (MCFP) and uses a fast algorithm based on Dijkstra's algorithm to solve it (section 6.3). Overall, while solving the LP does not scale well, our fast variant of the band method has reasonable performance on moderate size grids.

Coupling with solids is a fundamental problem, and we address it within our framework that maintains incompressibility (section 7). In our evaluation, we devised scenarios (section 8.1) that illustrate the advantage of our method over the state of the art. One aspect that we show is that gradual correction over time, which previous works have in common, may not be adequate in certain scenarios: There may not be an opportunity to correct the fluid after an obstacle moved, and guaranteeing incompressibility in each time step is therefore necessary.

2 RELATED WORK

Fluid simulation was introduced to computer graphics by Foster and Metaxas [1996]. It was popularized by Stam [1999], who took the Eulerian view, using a grid to discretize the fluid. A semi-Lagrangian approach that simulates a particle movement was used in the advection step to ensure unconditional stability of the simulation. Kim et al. [2007] simulate bubbles using the level set method. They track the volume change of each connected region and compensate for errors using divergence.

Smoothed-particle hydrodynamics (SPH) [Koschier et al. 2022] takes the Lagrangian view, representing fluids with particles. The approach was used in simple and intuitive methods such as [Macklin and Müller 2013; Müller et al. 2003], reminiscent of the boids algorithm [Reynolds 1987], where from local rules for a particle, based on its neighborhood, emerges a global behavior. Bender and Koschier [2017] combine two pressure solvers, one enforcing a divergence-free velocity field, the other satisfies a constant density condition. Band et al. [2018] improve the implicit incompressible SPH (IISPH) [Ihmsen et al. 2014] by realizing a consistent pressure gradient at boundary samples, using a different discretization of the pressure equation.

Hybrid schemes based on the particle-in-cell (PIC) and fluid-implicit-particle (FLIP) methods [Zhu and Bridson 2005] use a dual view combining grid and particles for fluid representation. The approach is similar to the material point method (MPM) [Jiang, Schroeder, Teran, et al. 2016], which was used to simulate a larger variety of materials including elasto-plastic constitutive models. Between the algorithms steps, data is transferred between the two representations, which causes a loss of information. A few methods aim at mitigating the loss. APIC [Jiang, Schroeder, Selle, et al. 2015] endows each particle with additional information in the form of a matrix, which allows it to preserve angular momentum. PolyPIC

[Fu et al. 2017] improves the energy and vorticity conservation of APIC by considering more velocity modes.

Ando, Thurey, et al. [2012] detect and preserve thin fluid sheets, which are reconstructed using anisotropic kernels. Um et al. [2014] use sub-grid particle correction for better particle distribution. The band method [Ferstl et al. 2016] keep particles only within a narrow band of the liquid surface to improve performance. Sato et al. [2018] extend the band method and add particle correction based on [Ando, Thurey, et al. 2012] to better distribute particles near the surface. Takahashi and Lin [2019] simulate viscous materials based on APIC with strong two-way coupling with solids. They apply position correction based on density constraints, using a purely Lagrangian approach (SPH).

From the mass conservation law, Kugelstadt et al. [2019] derive a pressure Poisson equation which takes density deviation into account. They add a density correction step that recovers fluid volume, which involves solving an additional Poisson equation. Density correction was previously performed when using the so-called unilateral incompressibility constraint, which was used for free-flowing granular materials [Narain et al. 2010] and animating splashing liquids [Gerszewski and Bargteil 2013].

Power particles [De Goes et al. 2015] considers particles as having volume, and the fluid domain is partitioned as a power diagram. The particle volumes can be prescribed, which enables controlling the fluid's volume and leads to better particle distribution. Power PIC [Qu et al. 2022] improves the performance of [De Goes et al. 2015] by reformulating the problem as a transportation problem, which is solved efficiently using Sinkhorn's iterative algorithm. In a 2-phase simulation, an estimate of the surface is used as an air occupancy baseline for slack air variables. The variables fill the gaps between the prescribed volume of the fluid particles and the volume of the cells.

Elcott et al. [2007] rewrite the Euler equations in terms of vorticity instead of pressure. They use discrete exterior calculus (DEC) for discretization, which is readily applied to meshes. DEC theory guarantees that certain properties hold, which arbitrary discretizations [Ando, Thürey, et al. 2013] cannot. Ando, Thuerey, et al. [2015] apply the same approach to a regular grid, which reduces DEC to finite difference operators. It coincides with the MAC grid discretization [Bridson 2015], which is justified by DEC theory that associates k-forms with specific elements of the grid. The equations are derived from an energy, using a variational approach [Batty et al. 2007]. De Witt et al. [2012] represent vorticity and velocity using a basis of Laplacian eigenfunctions, which admits closed-form solutions on simple domains. Since Laplacian eigenfunctions correspond with spatial scales of vorticity, basis coefficients can be seen as a discrete spectrum of vorticity.

3 BACKGROUND

We model a fluid in a domain $\Omega \subset \mathbb{R}^d$, $d \in \{2, 3\}$, using the Navier–Stokes equations, which can be written for inviscid, incompressible flows as:

$$\rho \frac{\mathrm{D}u}{\mathrm{D}t} = -\nabla p + f \tag{1a}$$

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

Algorithm 1: A time step of the FLIP method

---

1 Transfer velocity from particles to grid `// particles are at x̄`
2 Apply external forces to grid
3 Solve for pressure
4 Transfer velocity from grid to particles
5 Advect particles `// new positions are at x̂`
6 Correct particle positions `// new positions are at x`

---

where $u, f \in \mathbb{R}^d$ and $p, \rho \in \mathbb{R}$ denote velocity, external forces, pressure, and density. $\frac{D}{Dt}$ denotes the material derivative. Equation (1b) enforces the velocity to be divergence-free, which ensures incompressibility.

The FLIP method is a hybrid discretization method that combines the Eulerian and Lagrangian views. The domain is discretized using a regular (square or cubic) grid, and the fluid is discretized using particles. The FLIP algorithm alternates between the views, solving pressure over the grid, which is more accurate, and advecting quantities using particles, which is more robust. Discretization inaccuracies accumulate over time and violate incompressibility, and a correction step is needed to maintain it. The algorithm is listed in alg. 1 for a single time step. The optional correction step is missing in the original FLIP method, and it is implemented differently by IDP and our method.

4 DISCRETE INCOMPRESSIBILITY

We propose a definition for discrete incompressibility based on grid resolution. We define discrete density as the number of particles in a grid cell, and we denote its units by ppc, which stands for particles per cell. We initially propose the following simple condition for discrete incompressibility, which we will relax in section 4.1: keep a constant number $\mu \in \mathbb{Z}$ of particles in a fluid cell throughout the simulation. μ is given, and it is usually based on the initial fluid state.

Preserving the condition is done in the correction step that is described in the following, starting with notations. Let C be the set of grid cells that cover Ω. We associate markings with grid cells, which describe their characteristics. Each marking has its subset of cells that are marked with it. Initially, we use the disjoint subsets C_{empty}, C_{solid}, and C_{fluid} to mark cells that are empty, part of a solid, or contain fluid. For each cell c, we define a set γ_c of the indices of the particles that are in the cell.

Given a set of n particles and their positions $\hat{x} \in \Omega^n$ after advection, we would like to solve for new particle positions $x \in \Omega^n$ that are close to $\hat{x}$ but preserve incompressibility. We will refer to $\hat{x}$ as ideal positions. We define a cost function for closeness that penalizes the distance between two points $q, r \in \Omega$:

$$\sigma_{\text{obj}}(q, r) := \|q - r\|_2^2 .$$

This leads to the following problem:

$$\min_x \quad \sum_{j=1}^n \sigma_{\text{obj}}\left(x_j, \hat{x}_j\right) \tag{2a}$$

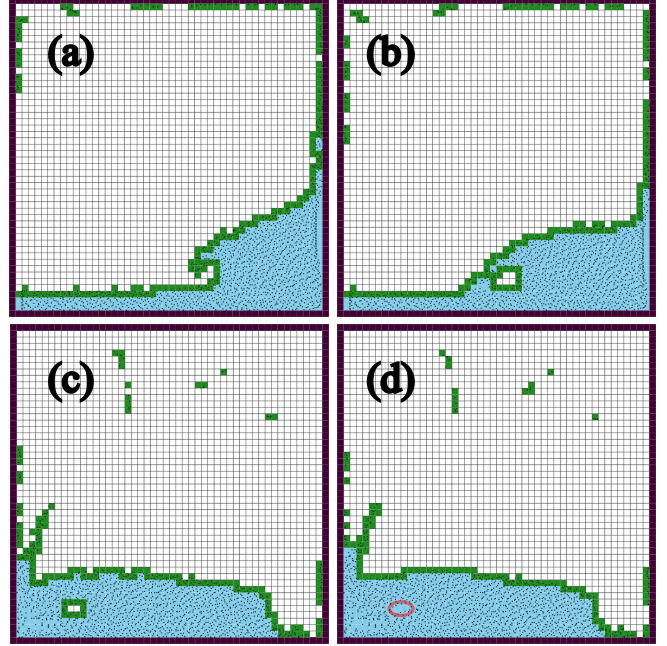$$\text{s.t.} \quad |\gamma_c| = \mu, \quad \forall c \in C_{\text{fluid}} \tag{2b}$$

Fig. 2. The surface (green cells) of a breaking wave (a) closes upon itself, creating an air pocket (b). The air pocket shrinks (c) until surface cells do not have empty neighbors anymore and become inner cells (d). These former surface cells may have less than μ particles, and in such a case we say that they contain air bubbles (an example is circled in red).

$$|\gamma_c| \le 0, \quad \forall c \in C_{\text{solid}}, \tag{2c}$$

where $x_j, \hat{x}_j \in \Omega$ are the new and ideal positions of the jth particle, and $|\gamma_c|$ denotes the number of particles in cell c. Cell markings C_{empty}, C_{fluid} and the sets γ are determined by x. That is, C_{solid} is determined by objects in the scene, and eq. (2c) ensures that it does not contain particles. C_{empty}, on the other hand, is determined by particle positions, and it does not require a constraint.

4.1 Surface and Bubbles

The discrete incompressibility condition in the previous section is too restrictive, and we relax it in a reasonable way.

We will use two types of neighborhoods for a grid cell. The first is a von Neumann neighborhood, which refers to 4-connectivity in 2D and 8-connectivity in 3D (axis-aligned directions). The second is a Moore neighborhood, which refers to 8-connectivity in 2D and 26-connectivity in 3D (two grid cells are neighbors if the Chebyshev distance between their centers is 1).

Definition 4.1 (surface). Given cell markings, we define a fluid cell as surface if it neighbors in a Moore neighborhood a cell that is not a fluid cell and is not on the domain boundary. We partition C_{fluid} into surface cells C_{surface} and inner cells C_{inner}.

In the definition, we used only the domain boundary, distinguishing between solid cells that are static (tank walls) and solid cells that may move (section 7).

We start with motivation. Consider a common setup where the fluid initially occupies a rectangular shape of 2D grid cells with density μ ppc. The incompressibility conditions from the previous sections means that the fluid will flow and change in full cells only, i.e., a cell from the surface with four particles will become empty, and an empty cell near the surface will gain four particles. This rigidity will cause particles to lose their resolution and behave as a unit or a single particle within a cell. To allow shape flexibility and individual particle movement between cells, we permit surface cells to have $\leq \mu$ particles. In the next section, we will extend the relaxation on surface cells to another layer of cells incident to the surface to make the problem easier.

Definition 4.2 (bubble). An inner cell $c \in C_{\text{inner}}$ is said to contain an air bubble if it is not empty and the number of its particles is less than μ.

In certain cases, such as a breaking wave, the surface curls and folds upon itself. This leads to a moment in time, where a surface cell suddenly becomes an inner cell since it no longer has any incident empty cells. This former surface cell may be only partially filled (has less than μ particles), which would violate the incompressibility condition.

To address this, we allow inner fluid cells to keep air bubbles. Specifically, instead of demanding from inner cells to have at least μ ppc, we demand that they will not lose particles (since becoming an inner cell) and will have at least as many particles as they had in the previous iteration. See fig. 2 for an illustration.

We apply the relaxed conditions to eq. (2):

$$\min_x \quad \sum_{j=1}^{n} \sigma_{\text{obj}}\left(x_j, \hat{x}_j\right) \tag{3a}$$

$$\text{s.t.} \qquad |\gamma_c| \leq \mu, \quad \forall c \in C_{\text{surface}} \tag{3b}$$

$$|\bar{\gamma}_c| \leq |\gamma_c| \leq \mu, \quad \forall c \in C_{\text{inner}} \tag{3c}$$

$$|\gamma_c| \leq 0, \quad \forall c \in C_{\text{solid}}, \tag{3d}$$

where $\bar{\gamma}$ refers to γ from the previous iteration. Note that while the relaxed constraints allow reasonable expansion, they do not allow compression: there can be at most μ ppc in a cell.

5 GRID MOVEMENT

The problem in eq. (3) is hard. To make it more manageable, we reformulate it in terms of grid movement.

We start by limiting the size of the simulation time step such that every particle does not move more than one cell (i.e., to the local Moore neighborhood). $\hat{x}$ will refer to particle positions after advection with the updated time step.

We reduce the possible grid movements of a particle to a set of grid directions in a von Neumann neighborhood. Namely, define the set $\mathcal{D}$ that consists of the columns of the $d \times d$ identity matrix and their negation, along with the zero vector that signifies staying in the same cell. For example, in 2D:

$$\mathcal{D} := \{\mathcal{D}_i\}_{i=1}^{5} = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}. \tag{4}$$
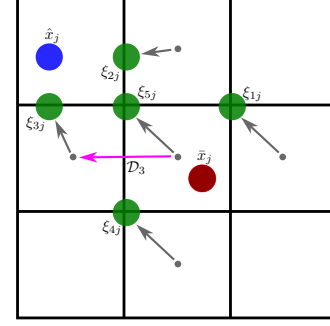

Fig. 3. Possible positions ξ_{ij} (in green) of the jth particle. A gray arrow points from the center of the cell that the particle is confined to to the closest location in that cell (up to a margin ϵ) to $\hat{x}_j$ (in blue). If $\hat{x}_j$ was in one of the five possible cells, then the corresponding ξ_{ij} to that cell would have coincided with it. All the possible positions ξ_{ij} are known, and a solution to eq. (6) selects one as the position x_j of the particle at the end of the iteration. There are five cells corresponding to $\mathcal{D}$. One example of a discrete direction is give by the magenta arrow, which points from the center of the cell that contains $\bar{x}_j$ (in red) to the center of the neighboring cell in the discrete direction $\mathcal{D}_3$.

Let $m := |\mathcal{D}|$ (m=5 in 2D, and m=7 in 3D). Experimentally, we found that using a Moore neighborhood (9 directions in 2D and 27 directions in 3D) did not make a significant difference.

Instead of solving for particle positions, we solve for particle grid movements. For each particle, we choose one direction from $\mathcal{D}$. Let $b \in \mathbb{Z}_2^{m \times n}$ be a binary matrix that chooses a direction for each particle. The jth column is assigned to the jth particle, and it contains a single nonzero in the entry that corresponds to the particle's chosen direction.

Let $\phi_{\text{center}}(c) \in \Omega$ return the center of a cell c. Let $\phi_{\text{close}}(q, c) \in \Omega$ return the point in a cell c that is closest to a point $q \in \Omega$. That is, the kth component of $\phi_{\text{close}}(q, c)$ is

$$\left(\phi_{\text{close}}(q, c)\right)_k := \begin{cases} q_k & \text{if } |v_k| < 0.5 \\ \lceil \left(\phi_{\text{center}}(c)\right)_k \rceil - \epsilon & \text{if } v_k \geq 0.5 \\ \lfloor \left(\phi_{\text{center}}(c)\right)_k \rfloor + \epsilon & \text{if } v_k \leq -0.5 \end{cases}, \tag{5}$$

where $v := q - \phi_{\text{center}}(c)$, and ϵ (=0.01) is a small margin.

Let $\bar{x} \in \Omega^n$ be the particle positions at the end of the previous iteration. The matrix entry b_{ij} corresponds to a possible grid movement $\mathcal{D}_i \in \mathbb{R}^d$ for the jth particle. We associate a specific particle position with this entry, which is the optimal position in the cell that the particle will end up in w.r.t. its ideal position and σ_{obj}:

$$\xi_{ij} := \phi_{\text{close}}\left(\hat{x}_j, \text{cell}\left(\bar{x}_j + \mathcal{D}_i\right)\right),$$

where $\bar{x}_j$ denotes the position of the jth particle in the previous iteration, and cell $(\cdot)$ returns the grid cell that contains a given point in the domain. Note that all positions $\xi \in \Omega^{m \times n}$ are known. See fig. 3 for illustration.

For each cell c, we define a set $\tilde{\gamma}_c$ of index pairs for the particles that may end up in it. Each index pair is associated with a particle movement possibility, and it consists of an index of a direction and

an index of a particle:

$$\tilde{\gamma}_c := \left\{ (i,j) \mid \text{cell}\left(\xi_{ij}\right) = c \right\}$$

The problem becomes:

$$\min_{b} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} b_{ij} \sigma_{\text{obj}}\left(\xi_{ij}, \hat{x}_j\right) \tag{6a}$$

$$\text{s.t.} \qquad 0 \le b_{ij} \le 1, \qquad \forall b_{ij} \in b \tag{6b}$$

$$\sum_{i=1}^{m} b_{ij} = 1, \qquad j = 1 \ldots n \tag{6c}$$

$$\sum_{(i,j) \in \tilde{\gamma}_c} b_{ij} \le \mu, \qquad \forall c \in \bar{C}_{\text{empty}} \cup \bar{C}_{\text{surface}} \tag{6d}$$

$$|\bar{\gamma}_c| \le \sum_{(i,j) \in \tilde{\gamma}_c} b_{ij} \le \mu, \qquad \forall c \in \bar{C}_{\text{inner}} \tag{6e}$$

$$\sum_{(i,j) \in \tilde{\gamma}_c} b_{ij} \le 0, \qquad \forall c \in \bar{C}_{\text{solid}}. \tag{6f}$$

Details:

- The objective in eq. (6a) is similar to eq. (3a). All $\sigma_{\text{obj}}\left(\xi_{ij}, \hat{x}_j\right)$ are known, and b ensures that only selected particle movements contribute to the sum.
- Equation (6b) asserts the range of binary variables.
- Equation (6c) forces a single selected direction for each particle.
- Equation (6d) and eq. (6e) are similar to the incompressibility constraints eqs. (3b) to (3c). The sum $\sum_{(i,j) \in \tilde{\gamma}_c} b_{ij}$ counts the particles that end up in cell c. $\bar{C}$ refers to the markings in the previous iteration.
- Equation (6f) is similar to eq. (3d)

In eq. (6d), we use the surface marking from the previous iteration to relieve the need to track the surface during optimization (or formulate a constraint that handles the two cases of a surface cell remains a surface or becomes an inner cell). This extends the relaxed condition on the surface from section 4.1 to another layer of cells incident to the surface (the condition now applies to surface cells in the previous iteration, which may belong to the layer of inner cells incident to the surface in this iteration), which is still within reason.

The problem is always feasible since $\bar{x}$ is in the solution space. Given a solution b^*, the final particle position x_j is set to the ξ_{ij} that corresponds to its selected movement direction indicated by the ith entry with the single nonzero in the jth column of the solution b^*.

The problem in eq. (6) is a linear programming problem with binary variables b only (the rest of the symbols are fixed, including cell markings, index sets, and particle positions, which do not depend on b), which is a type of integer linear programming (ILP). Satisfying a 0-1 ILP is one of Karp's 21 NP-complete problems. The following proposition allows us to relax the problem to a standard linear programming (LP) with continuous variables $b \in \mathbb{R}^{m \times n}$, for which there are polynomial-time solvers, and it is faster to solve in general.

PROPOSITION 5.1. *The LP relaxation of the ILP in eq.* (6), *which uses continuous variables, has the same optimal solution.*
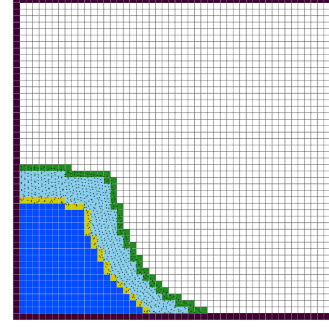
See proof in appendix A.

Fig. 4. A band. Deep cells ($C_{<-R}$) in dark blue, band interface (C_{-R}) in yellow, surface (C_0) in green, and the rest of the band ($C_{-R<\beta<0}$) in light blue.

6 THE BAND METHOD

Solving the LP in eq. (6) does not scale well, and for large 3D grids, we propose an incompressible variant of the band method [Ferstl et al. 2016]. The method uses only a fraction of the number of particles, which directly affects the size of the LP.

The motivation for the band method is based on the observation that most of the interesting, complex behavior of a fluid happens close to the surface. FLIP uses particles to reduce numerical dissipation and keep the simulation lively. Based on the observation above, particles at deeper levels of the fluid do not contribute much to visual appearance. Leveraging that, the method maintains only a narrow band of particles near the fluid surface and uses an Eulerian-grid approach to simulate the rest of the fluid. The grid velocity at each iteration is determined by a combination of the two.

To maintain fluid density for incompressibility, we need to supervise the number of particles that enter and leave the band. Furthermore, while the method in [Ferstl et al. 2016] uses an approximate distance from the surface to define the band, our discrete approach that uses hard constraints requires a more careful estimate.

Definition 6.1 (depth). Each fluid cell is assigned a depth $\beta \in \mathbb{Z}$ that represents its discrete (signed) distance from the surface, and it is derived from the state of the fluid (particle positions) at a time step. The depth is assigned recursively in a breadth-first-search manner. Surface cells are assigned depth $\beta = 0$. Their neighboring fluid cells in a von Neumann neighborhood are assigned $\beta = -1$. The unassigned fluid neighbors of the $\beta = -1$ cells are assigned one level lower, $\beta = -2$, and so on until all fluid cells are assigned a depth. All non-fluid cell are assigned an arbitrary positive number (e.g., 1) as depth.

Let $R + 1 \in \mathbb{N}$ be the thickness of the particle band. We define the cells at depth $-R \le \beta \le 0$ to be within the band. We call cells at depth $\beta = -R$ *band interface* and cells at depth $\beta < -R$ *deep*. We add marking subsets to distinguish between parts and depth levels of the fluid. Denote by $C_{\beta=k}$ or simply C_k (when clear from the context) the set of cells at depth $\beta = k$. We extend the notation to a range of depth levels, e.g., $C_{<-R}$ will denote deep cells. See fig. 4 for illustration.

Algorithm 2: A time step of the band method

---

1 Transfer velocity from particles to grid **and combine with current grid velocity** // $\bar{x}$
2 Apply external forces to grid
3 Solve for pressure
4 Transfer velocity from grid to particles
5 **Advect grid velocity**
6 Advect particles // $\hat{x}$
7 Correct particle positions // x
8 **Update cell markings**
9 **Remove particles that reached the deep and add excess particles to the band interface**

Alg. 2 outlines the steps of the band method. Changes from alg. 1 are emphasized. When transferring velocity from particles to the grid, the particles' velocity is copied only for cells within the band, not including the band interface. The velocity in the rest of the grid cells remain unchanged. When correcting the particles position, we modify our algorithm to handle the band (next sections). Advecting grid velocity, which is needed for the part of the fluid without particles (not in the band), is done using the common semi-Lagrangian approach [Stam 1999]. Cell markings C are updated, and the step is emphasized in alg. 2 to clarify that it is performed at the end of the correction step and before removing and adding particles.

Particles are limited to the band, and particles which go deep are deleted. To maintain incompressibility, the excess of particles in the deep is moved into the band interface, as described next. We keep track in a variable n_{deep} of the number of (imaginary) particles that are in the deep, updating the variable every deletion and insertion of a particle. The excess of particles in the deep is

$$n_{\text{excess}} := n_{\text{deep}} - \mu|C_{<-R}| \,.$$

When $n_{\text{excess}} > 0$, we add n_{excess} particles to the band interface. We randomly iterate the cells in the band interface and fill them up to μ with remaining excess particles. Each added particle is positioned randomly within a cell, and its velocity is interpolated from the grid velocity. Note that there is always space in the band interface for excess particles from the deep since we constrain the number of movements into and out of the band interface (next sections).

In the next sections, we offer three variants to control the movements into and out of the band interface, where each is faster than the former. Foundation and concepts are laid out through the sections, culminating in the fastest variant.

6.1 A Band Constraint

We maintain incompressibility by controlling the comings and goings of particles through the band interface. We want the number of particles that move from a shallower depth level ($\beta = 1 - R$) to the band interface ($\beta = -R$) to be equal to the number of particles that move in the other direction.

We define two sets of index pairs of particle movement possibilities, into and out of the band interface (from and into a shallower

level):

$$\tilde{\gamma}_{\text{in}} := \left\{ (i,j) \mid \text{cell}\left(\bar{x}_j\right) \in \bar{C}_{1-R} \,,\; \text{cell}\left(\xi_{ij}\right) \in \bar{C}_{-R} \right\}$$

$$\tilde{\gamma}_{\text{out}} := \left\{ (i,j) \mid \text{cell}\left(\bar{x}_j\right) \in \bar{C}_{-R} \,,\; \text{cell}\left(\xi_{ij}\right) \in \bar{C}_{1-R} \right\} \,.$$

The deep may contain air bubbles from cells that carried bubbles while moving to the deep. We would like to allow bubbles in the band interface and deep to fill up. Let α_{-R}, $\alpha_{<-R}$ be the total amounts of air bubbles (number of missing particles) in the band interface and deep, which can be calculated from the number of cells and particles:

$$\alpha_{-R} := \mu|\bar{C}_{-R}| - \sum_{c \in \bar{C}_{-R}} |\bar{\gamma}_c|$$

$$\alpha_{<-R} := \mu|\bar{C}_{<-R}| - n_{\text{deep}} \,.$$

We express the conditions above as an additional constraint to eq. (6):

$$0 \leq \sum_{(i,j) \in \tilde{\gamma}_{\text{in}}} b_{ij} - \sum_{(i,j) \in \tilde{\gamma}_{\text{out}}} b_{ij} \leq \alpha_{\leq-R} \,. \tag{7}$$

Where $\alpha_{\leq-R} := \alpha_{-R} + \alpha_{<-R}$. We also update eq. (6d) and eq. (6e) to use the band markings:

$$\sum_{(i,j) \in \tilde{\gamma}_c} b_{ij} \leq \mu \,, \quad \forall c \in \bar{C}_{\text{empty}} \cup \bar{C}_{\text{surface}} \cup \bar{C}_{-R} \tag{8a}$$

$$|\bar{\gamma}_c| \leq \sum_{(i,j) \in \tilde{\gamma}_c} b_{ij} \leq \mu \,, \quad \forall c \in \bar{C}_{-R<\beta<0} \,, \tag{8b}$$

where we allow the band interface the same flexibility as the surface (to lose particles) since excess deep particles will be added back to it. We put no constraint on deep cells due to the particle deletion step.

The correction step is done as before by solving the updated problem in eq. (6) for b and updating x accordingly. Unlike the local, sparse constraints in eq. (6), the band constraint is global and dense since it encompasses and ties together particle movements along the band interface. Moreover, the system matrix may not be totally unimodular anymore, and the ILP problem cannot be relaxed. In some scenes, these increase the solver time such that it is not much better than not using a band ($R = \infty$). In the next sections, we offer faster alternatives.

6.2 A One-Way Band Constraint

One way to shorten the constraint in eq. (7) is to determine first the number of particles that go into and out of the band, and based on that, constrain only the number of particles in the direction with the greater flow. We do this in two steps, solving an LP in the first step and an ILP in the second.

First, we solve eq. (6) as is (without an additional band constraint), getting an optimal solution b^*, and we do not update x yet. From these particle movements, denote the number of particles that go into and out of the band interface by

$$n_{\text{in}}^* := \sum_{(i,j) \in \tilde{\gamma}_{\text{in}}} b_{ij}^*$$

$$n_{\text{out}}^* := \sum_{(i,j) \in \tilde{\gamma}_{\text{out}}} b_{ij}^* \, .$$

Consider the differences

$$s_{\text{in}} := n_{\text{out}}^* - n_{\text{in}}^* + \alpha_{\leq -R}$$
$$s_{\text{out}} := n_{\text{in}}^* - n_{\text{out}}^* \, .$$

s_{in} measures how much space is left in the band interface and deep, and s_{out} measures the space in the rest of the band. If both $s_{\text{in}} \geq 0$ and $s_{\text{out}} \geq 0$, then the movements are fine, we can update x according to b^* and proceed with the rest of the algorithm. Else, there is negative space (incompressibility is violated), and we solve eq. (6) a second time with an additional constraint, depending on which space is negative.

If $s_{\text{in}} < 0$, then too many particles moved into the band interface, and we need to limit them. We fix the movements of all n_{out}^* particles that moved from the band interface to the rest of the band and block the rest of the movements in $\tilde{\gamma}_{\text{out}}$. In addition, we prevent movement into the band from particles in $\bar{C}_{1-R}$ that do not move into the band interface in b^*. We end up with a constraint that selects $n_{\text{out}}^* + \alpha_{\leq -R}$ particles from the particles that moved into the band in b^*:

$$b_{ij} = b_{ij}^* \, , \qquad \forall \, (i,j) \in \tilde{\gamma}_{\text{out}} \qquad (9a)$$

$$b_{ij} = 0 \, , \qquad \forall \, (i,j) \in \tilde{\gamma}_{\text{in}} \, , \ b_{ij}^* = 0 \qquad (9b)$$

$$\sum_{(i,j) \in \tilde{\gamma}_{\text{in}}} b_{ij} = n_{\text{out}}^* + \alpha_{\leq -R} \, . \qquad (9c)$$

When setting a movement b_{ij} of the jth particle in eq. (9a) to one, due to eq. (6c), we can also set the rest of the particle's movements to zero: $\forall k \neq i, b_{kj} = 0$. Note that eq. (9a) and eq. (9b) merely eliminate variables from the system, which leaves a single constraint eq. (9c) that sets the number of particles that enter the band interface. Due to b^*, which moves more particles than required, we know that the problem in eq. (6) with the additional constraint eq. (9) is feasible.

Else, $s_{\text{out}} < 0$, and we need to limit the number of particles that move out of the band. Similar to eq. (9), this is expressed as

$$b_{ij} = b_{ij}^* \, , \qquad \forall \, (i,j) \in \tilde{\gamma}_{\text{in}} \qquad (10a)$$

$$b_{ij} = 0 \, , \qquad \forall \, (i,j) \in \tilde{\gamma}_{\text{out}} \, , \ b_{ij}^* = 0 \qquad (10b)$$

$$\sum_{(i,j) \in \tilde{\gamma}_{\text{out}}} b_{ij} = n_{\text{in}}^* \, , \qquad (10c)$$

where eq. (10a) fixes the variables of movements into the band interface, eq. (10b) prevents movements out of the band interface that do not occur in b^*, and eq. (10c) sets the number of particles that leave the band interface

To summarize, in the first step we solve eq. (6). If needed, we perform a second step, where we solve eq. (6) again using the one-way band constraints in eq. (9) or eq. (10). After the steps, we update x and proceed with the rest of the algorithm. See fig. 5 for an example.

The one-way band constraint is still dense, and the ILP still cannot be relaxed. However, since we reduce variables and simplify the band problem, it becomes significantly faster to solve than adding the full band constraint in eq. (7).

6.3 Flow Along Paths

The second step of the one-way band constraint approach can be viewed as correcting the incompressibility in the band interface and the deep after the first step. We suggest a cheaper way to perform the correction, which does not require solving an ILP.

We perform the same first step as in section 6.2 and solve eq. (6) as is (an LP without additional band constraints), this time updating the particles positions x according to b^*. If s_{in} or s_{out} is negative, then too many particles flowed into or out of the band interface. To correct that, we move some of them along grid paths in the required direction.

If s_{in} is negative, then we need to move $n_{\text{move}} := -s_{\text{in}}$ particles out of the band interface. Else, if s_{out} is negative, then we need to move $n_{\text{move}} := -s_{\text{out}}$ particles into the band interface. Otherwise, correction is not necessary.

We limit the jth particle's movement to a single cell (in a von Neumann neighborhood) relative to its position in the last iteration ($\bar{x}_j$). To maintain the incompressibility constraint, a particle can move into cell c only if it has space ($|\gamma_c| < \mu$, where γ reflects the state of the updated x). If it does not, then another particle needs to move out from c beforehand. This means that a chain of particles needs to be moved along a grid path, starting from a cell that has the flexibility to lose a particle—a surface, a band interface, or a former empty cell (see eq. (8a)). We need to find n_{move} such paths.

We formulate this as a minimum-cost flow problem (MCFP) in a graph. The grid cells are designated as graph vertices, and possible particle movements are designated as graph edges with capacity one. We will use multiple sources and sinks, denoted C_{source} and C_{sink}. The location of sources and sinks depend on the flow direction—into or out of the band interface. In the in direction, surface cells are sources, and band interface and deep cells are sinks. In the out direction, interface cells are sources, and the rest of the band (surface and inner cells with bubbles) and empty cells are sinks.

The cost of an edge that represents a possible movement of the jth particle into a cell c is the cost of its (optimal) position in c minus the cost of its current position w.r.t. its ideal position:

$$\sigma_{\text{edge}}(j,c) := \sigma_{\text{obj}}\left(\phi_{\text{close}}\left(\hat{x}_j, c\right), \hat{x}_j\right) - \sigma_{\text{obj}}\left(x_j, \hat{x}_j\right) \, . \quad (11)$$

We limit the movement of the jth particle to its cell in the previous iteration and the cells neighboring that cell (in a von Neumann neighborhood).

To solve the MCFP, we use a variant of Dijkstra's algorithm to find n_{move} (augmenting) paths in the (residual) graph (using terms from the Ford–Fulkerson algorithm). The algorithm is listed in alg. 3.

A path starts in a source cell and ends in a sink. The algorithm finds paths that do not share cells (or particles). We hold in three arrays (of size $|C|$), J, b_{fin}, and σ, data related to the cells. $J[c]$ is the index of a particle that represents an edge to the parent of c on a path, where c can belong to a single path at most. $b_{fin}[c]$ is a (boolean) flag that indicates if the path that started at the source cell c is complete. $\sigma[c]$ is the total cost of the path that c belongs to from its source to c. The three arrays are initialized with none, false, and ∞ (using multiple assignment in line 2).

The search for paths starts at source cells (skipping empty ones), which are pushed into a priority queue Q of objects of type *Node*
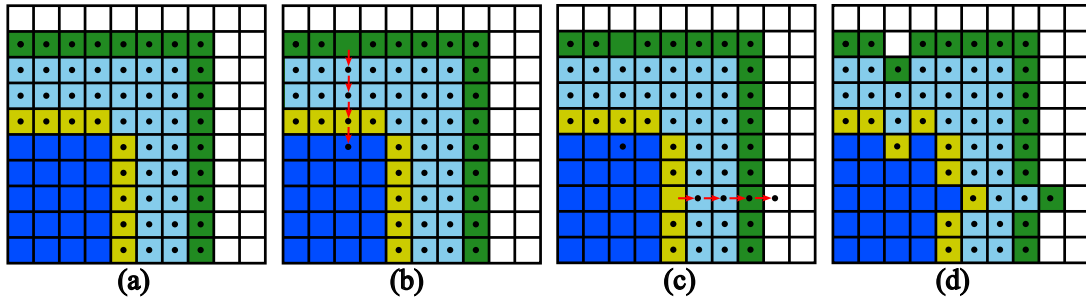
Fig. 5. An example of correcting the band (1 ppc). (a) The beginning of the iteration; see fig. 4 for the color code of the cells. (b) First step, solving the LP in eq. (6). Four particles move into the cell below them; red arrows indicate former cells. One of the particles moved into the band interface: $n_{\text{in}}^* = 1$, $n_{\text{out}}^* = 0$, $s_{\text{in}} = -1$, $s_{\text{out}} = 1$. Since $s_{\text{in}} < 0$, incompressibility is violated, and we need to perform a second step to correct it. If we use the variant in section 6.2, then we solve eq. (6) again—based on the particle positions in (a)—fixing the movements in eqs. (9a) to (9b) and adding the constraint in *eq.* (9c) to set the number of particles that move into the band (zero). There will be no movement of particles between cells (only within the cells) compared to (a); such movement will occur only when particles also leave the band interface in the first step. (c) If we use the variant of the band method in section 6.3, then a path from the band interface to the surface is found. Reverting the vertical path in (b) is always an option. Instead, the indicated horizontal path is selected, and particles are pushed along it. (d) Cell markings are updated. In this case, there is no need to remove particles that reached the deep or to fill the band interface with excess particles from the deep (alg. 2).

(line 6). A *Node* represents the last vertex in a path, which will probe for the next cell on the path. The fields of *Node* are: *cost*—the sum of edge costs along the node's path; *cell*—the cell that the vertex represents; *edge*—the index of a particle that represents an edge to the cell's parent; and *root*—the path's root (a source cell). A new *Node* is created using a constructor function with named arguments, and it is added to Q, where the field *cost* is used as a key to compare elements. The special value ROOT is used to indicate a root node (no parent).

In the main loop, the *Node* with the lowest cost is dequeued. Line 11 checks if the cell has already been visited, i.e., if it has been dequeued already and has been assigned a parent. Q can hold *Nodes* of the same cell but different parents (different possible paths). Only the *Node* with the lowest cost is processed, and the rest are ignored. This is an efficient alternative for the priority queue's decrease-key method for sparse graphs. If a path that started at the node's root has already finished, then the node is ignored. If it is the first time that the node is visited, then it is assigned an edge to the parent on that path or a ROOT tag (line 13). If the node is a sink with space (line 14), then the path is complete, and it is added to a (returned) list of completed paths P.

The neighboring cells of a node that were not visited yet are explored (line 18). The best edge to a neighboring cell c' is determined, and if the path to it has a lower cost, then c' is enqueued. Alg. 4 lists the algorithm that finds the best edge from cell c to c'. The condition in line 3 checks that c' is not farther than one cell from the particle's cell in the previous iteration. It means that either the particle is currently in the same cell as in the last iteration or it is going to move to that cell.

The paths are updated one at the time using alg. 5 until n_{move} paths are successfully updated.

Since alg. 3 finds only nonintersecting paths, it may need be called more than once (with the paths updated). Paths can be found (in the residual graph after an update) as long as the maximum flow is not reached. The maximum flow is at least n_{move} since it is possible to

revert the particle positions induced by b^* back to $\bar{x}$. However, since the edge costs may be negative and we use Dijkstra, the resulting flow from the algorithm may not have the lowest cost. We decided not to use a more expensive algorithm that finds the optimal cost since n_{move} is only a small percentage of n, and in our experiments the results of using Dijkstra did not vary much from the alternative methods suggested in the previous sections.

See fig. 5 for an example.

7 COUPLING WITH SOLIDS

We address incorporating solids into our framework. We illustrate the idea on a simple scene of an object (also referred to as obstacle) free falling into water. Before the object hits the water, its motion is only affected by gravity. After hitting the water, the drag and buoyancy forces come into play, which decelerate the object until it reaches terminal velocity. The affect on the fluid is expressed in the boundary conditions of the pressure equation [Bridson 2015, chapter 5]. The pressure in grid cells that are marked as solid is set to $p = 0$, and along the solid boundary we have

$$u \cdot n = u_{\text{solid}} \cdot n \,, \tag{12}$$

where n is the boundary normal, and u_{solid} is the solid velocity.

We discretize an object using particles, but we will keep using the term particles only to refer to fluid particles. We allow a (non-empty) grid cell to be occupied by either fluid or a solid, which determines its marking. As we did with fluid particles, the time step size is limited to restrict the solid from moving past the neighboring cells (Moore neighborhood). If the object is going to occupy new cells, then the correction step decides whether the object moves or stays in place. We consider objects that are denser than the fluid, and the movement of the object is prioritized unless incompressibility or fluid speed are violated.

Let $C_{\text{new\_solid}}$ be the set of cells that are not marked as solid and the object intends to move into. We modify the objective in eq. (6a)

Algorithm 3: Find paths

Output: A list P of Paths and an array J of $|C|$ edges to parents

1   Let b_{fin} be an array of $|C|$ flags, σ be an array of $|C|$ costs, and Q be a priority queue

2   **for** $c \in C$ **do** $J[c], b_{fin}[c], \sigma[c] \leftarrow$ none, false, ∞

3   **for** $c \in C_{\text{source}}$ **do** // initialize Q

4   **if** $|\gamma_c| = 0$ **then** continue // no particles

5     $a \leftarrow$ Node(cost=0, cell=c, edge=ROOT, root=c)

6   Q.enqueue(a) // using a.cost as key

7     $\sigma[c] \leftarrow 0$

8   **while** not Q.empty() **do** // main loop

9     $a \leftarrow Q$.dequeue() // lowest cost

10     $c \leftarrow a$.cell

11   **if** $J[c] \notin \{$none, ROOT$\}$ **then** continue // visited

12   **if** $b_{fin}[a.\text{root}]$ **then** continue // finished

13     $J[c] \leftarrow a$.edge // assign an edge to parent

14   **if** $c \in C_{\text{sink}}$ and $|\gamma_c| < \mu$ **then** // a sink with space

15       $b_{fin}[a.\text{root}] = $ true

16   P.add(Path(edge=a.edge, sink=c))

17   continue

18   **for** $d \in \mathcal{D} \setminus \{0\}$ **do** // without the 0 vector

19       $c' \leftarrow$ cell $(\phi_{\text{center}}(c) + d)$

20   **if** $J[c'] \neq$ none **then** continue // visited or a source

21       $j \leftarrow$ best_edge(c, c')

22   **if** $j =$ none **then** continue // no edge

23       $t \leftarrow \sigma[c] + \sigma_{\text{edge}}(j, c')$ // total cost from source

24   **if** $t \geq \sigma[c']$ **then** continue // is not better

25       $\sigma[c'] \leftarrow t$

26   Q.enqueue(cost=t, cell=c', edge=j, root=$a.\text{root}$))

Algorithm 4: best_edge(c, c')

Input: Two cells c, c'

Output: An index j of a particle that can move from c to c' with the lowest cost

1   $\sigma, j \leftarrow \infty$, none

2   **for** $j' \in \gamma_c$ **do**

3   **if** $c \neq$ cell $(\bar{x}_{j'})$ and $c' \neq$ cell $(\bar{x}_{j'})$ **then** continue

     // farther than one cell from $\bar{x}_{j'}$

4   **if** $j =$ none or $\sigma_{\text{edge}}(j', c') < \sigma$ **then**

     $\sigma, j \leftarrow \sigma_{\text{edge}}(j', c'), j'$

---

to use a new objective function:

$$\min_b \quad \sum_{i=1}^{m} \sum_{j=1}^{n} b_{ij} \sigma_{\text{solid\_obj}} \left(\xi_{ij}, \hat{x}_j \right) , \tag{13}$$

Algorithm 5: Update a path

Input: An array J of $|C|$ edges to parents and a Path r

1   $j, c \leftarrow r$.edge, r.sink

2   **while** $j \neq$ ROOT **do**

3     $c' \leftarrow$ cell (x_j) // parent cell

4     $x_j \leftarrow \phi_{\text{close}}(\hat{x}_j, c)$ // move particle

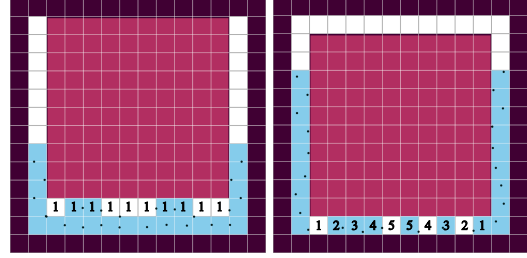5 $j, c \leftarrow J[c'], c'$ // predecessor

---

Fig. 6. Clearing distance. Two frames of an obstacle (red square) falling into water (1 ppc). The clearing distance of cells in $C_{\text{new\_solid}}$ is marked. (Left) all the cells in $C_{\text{new\_solid}}$ have a non-obstacle neighbor below them, and their clearing distance is 1. (Right) Only the particle in the rightmost cell in $C_{\text{new\_solid}}$ is guaranteed to clear the way using eq. (14). Other particles in $C_{\text{new\_solid}}$ require eq. (15) to guarantee progress towards clearing the way.

where

$$\sigma_{\text{solid\_obj}}(q, r) := \begin{cases} \lambda_{\text{penalty}} & \text{if cell}(q) \in C_{\text{new\_solid}} \\ \sigma_{\text{obj}}(q, r) & \text{else} \end{cases} . \tag{14}$$

λ_{penalty} (=1000) is set to a large weight to penalize particle movements into (potentially) new solid cells.

Given a solution to the modified problem, the correction step lets the object move only if none of the new particle positions x are in $C_{\text{new\_solid}}$; else the object stays in place.

For the band method in section 6.3, we modify σ_{edge} in eq. (11) to use $\sigma_{\text{solid\_obj}}$ instead of σ_{obj}.

The definition of the fluid surface (definition 4.1) considers fluid cells that touch a moving obstacle as surface. This enables flexibility in the movement of an obstacle.

7.1 Clearing the Bottom

Consider particles that occupy potentially new obstacle cells and currently block the obstacle movement. As long as there is a path in the fluid to a cell ($\notin C_{\text{new\_solid}}$) with free space, an optimal solution will push them along the path to move them out of the obstacle's way and avoid the penalty in eq. (14).

There is always such a path in the fluid (as long as there is space) except when the obstacle reaches the last layer of fluid before touching the bottom of the tank. Consider such a row of cells in $C_{\text{new\_solid}}$ between the obstacle and the bottom, where all the cells are empty except the middle one, which contains a particle. The particle has three possible cell movements: stay in the current cell, go left, or

go right. Its contribution to the objective in eq. (13) would be the same in each case, and nothing would motivate it to clear the way. To address that, similar to definition 6.1, we define

Definition 7.1 (clearing distance). Each cell $C_{\text{new\_solid}}$ is assigned a clearing distance that represents its discrete distance from a cell that is not in $C_{\text{new\_solid}}$, where solid cells are ignored. The clearing distance is assigned recursively in a breadth-first-search manner. Cells in $C_{\text{new\_solid}}$ with neighboring cells (in a von Neumann neighborhood) that are not in $C_{\text{new\_solid}}$ are assigned 1. Their unassigned neighbors are assigned one level higher, 2, and so on until all the cells in $C_{\text{new\_solid}}$ are assigned a clearing distance.

We modify eq. (14):

$$\sigma_{\text{solid\_obj}}\left(q, r\right) := \begin{cases} \lambda_{\text{penalty}} \cdot \text{cdist}\left(q\right) & \text{if cell}\left(q\right) \in C_{\text{new\_solid}} \\ \sigma_{\text{obj}}\left(q, r\right) & \text{else} \end{cases},$$

(15)

where $\text{cdist}\left(\cdot\right)$ returns the clearing distance at cell $\left(q\right)$. This penalizes particles according to their clearing distance; see fig. 6 for illustration.

8 EVALUATION

We implemented our method as a plugin in MantaFlow [Thuerey and Pfaff 2018], using conjugate gradients to solve a Poisson equation. We used [Gurobi 2018], selecting the dual simplex algorithm without presolve, to solve LP and ILP problems.

Measuring running time. The experiments were conducted on a laptop. The running time of FLIP and IDP is dominated by solving a Poisson equation. FLIP solves one for pressure, and IDP solves an additional one for density. The running time of our method is dominated by the solution of the LP problem.

Volume measure. We define the discrete volume measure of a cell c based on its depth (definition 6.1) as

$$V_c := \begin{cases} 0 & c \in C_{\beta > 0} \\ \min\left(1, \frac{\|\gamma_c\|}{\mu}\right) & c \in C_{-1 \leq \beta \leq 0} \\ 1 & \text{else} \end{cases}.$$

(16)

Cells near the surface are given reasonable flexibility and are allowed to have less than μ particles. Other fluid cells are penalized if they have less than μ particles. All cells are penalized if they have more than μ particles. Solid cells that contain particles are still considered pure solid, and their fluid volume is zero.

The measure used in [Kugelstadt et al. 2019] is $\min\left(1, \frac{\|\gamma_c\|}{\mu}\right)$ for any cell c. That measure is more favorable towards our method since it penalizes overflow only and overlooks air bubbles (volume inflation). According to that measure, our method preserves discrete volume perfectly.

When reporting results, we measure the volume of the whole fluid in a time step as $\frac{V}{V^*}$, where $V := \sum_{c \in C} V_c$ is the total fluid volume in a time step, and V^* is how much volume should the fluid occupy. If there is no emitter in the scene, then V^* is simply the initial fluid volume. We report the range of volume percentages
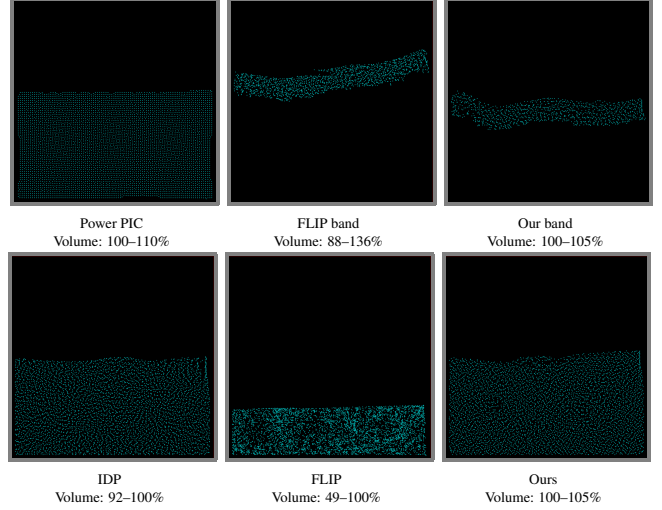

Fig. 7. The last frame in a dam scene.

$\left(100 \frac{V}{V^*}\right)$ over all the simulation iterations.

We evaluated our method in several scenes that are described in section 8.1; see the accompanying video for their animation. Statistics on volume preservation and running time are summarized in table 1 and table 2. The grid sizes that were used in the figures and video are the ones in table 1. We compared our method with IDP [Kugelstadt et al. 2019], FLIP, the narrow band FLIP [Ferstl et al. 2016], and Power PIC [Qu et al. 2022].

Power PIC has several parameters that can be crucial for its behavior, the accuracy of its particle distribution, and volume preservation. We scaled the resolution of the transportation grid by 2 in each dimension (i.e., ×4 finer than the simulation grid in 2D). We set $\epsilon = 0.1$, $\eta = 1$, $\tau = \frac{1}{2\sqrt[4]{\mu}}$ (e.g., $\frac{1}{4}$ for 4 ppc in 2D), and $\delta = 0.1$. We did not cut off small coefficients from the Gaussian kernel K since it increased the number of iterations due to lower accuracy. Besides increasing the running time, large scaling of the transportation grid resulted in cracks and holes that kept forming and mending in the fluid. Larger values for ϵ and δ disrupted the uniform particle distribution. On the other hand, effects of using smaller values ranged from the fluid becoming sluggish, exhibiting extremely high energy dissipation, up to standing still. To summarize, Power PIC can correct the fluid's volume and particle distribution but at the risk of introducing dissipation if the changes are aggressive.

8.1 Scenes

The default settings that were used in the scenes (unless specified otherwise):

- The initial density is 4 ppc in 2D and 8 ppc in 3D.
- The band method is used for our method in 3D (the variant in section 6.3) and only for it. The band thickness is $R = 3$.
- The maximum fluid speed is bounded.

A breaking dam. In this scene, we perform an initial comparison of the methods behavior and volume preservation. Some time after
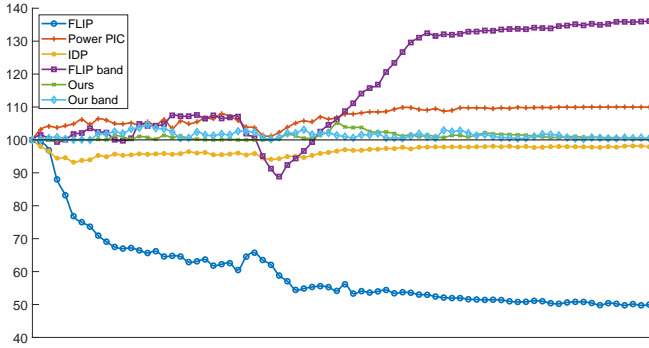
Fig. 8. Volume (in percent) over time of the 2D dam scene.

the dam breaks, an emitter spews a stream of water into the tank. After the emitter finishes, the total number of particles should fill exactly a half of the tank (domain); see fig. 7.

The volume is plotted in fig. 8. FLIP loses volume, and it is its general tendency. The volume of the narrow band FLIP fluctuates. Power PIC gives a nice distribution of particles and tends to preserve volume but not perfectly. IDP tends to preserve volume but suffers some compression. Our method uses constrained optimization and cannot lose volume. The volume may increase, however, due to air bubbles. Our band method behaves similarly. For both band methods we used thickness $R = 6$ due to the more lively behavior of the particles compared to other scenes.

Figure 9 shows a frame, where IDP keeps the clumped lines of particles and suffers volume loss. Power PIC and our method distribute the particles, which adds noise to the fluid that reaches the surface.

Figure 10 shows another 2D dam scene (without an emitter) using 1 ppc in a ×4-finer grid (i.e., scaled by two in each dimension). For FLIP and IDP, the fluid collapses, and there is a dramatic volume loss. This is due to the sparse particle distribution (1 ppc), where particles can easily clump together, and some cells are missed. As a result, the fluid is riddled with holes (see fig. 11). The holes have zero pressure, and they disrupt the velocity field and attract particles. Power PIC works hard to maintain a uniform distribution of the particles, and it requires significantly more Sinkhorn iterations for a time step. While its volume loss is less severe, the general behavior of the fluid is similar to FLIP. Our method is the only one to maintain reasonable fluid volume and behavior, which is similar to the 4 ppc case, and the rare occurrences of holes in the fluid do not disrupt the velocity field. Since there is only one particle at most in a grid cell, there can be no air bubbles, and the volume is perfectly preserved.

Figure 12 shows a 3D dam scene. FLIP loses a significant amount of volume. IDP preserves the volume but suffers some compression. IDP keeps the fluid smooth while ours introduces noise similar to the 2D case (fig. 9). Using our method, the fluid hits the left wall the same time FLIP does. IDP overshoots the splash, which hits the wall earlier and more strongly. The 1 ppc version of our method requires surface extraction of lower resolution, which is less detailed. Our 8 ppc version gains some volume due to air bubbles.
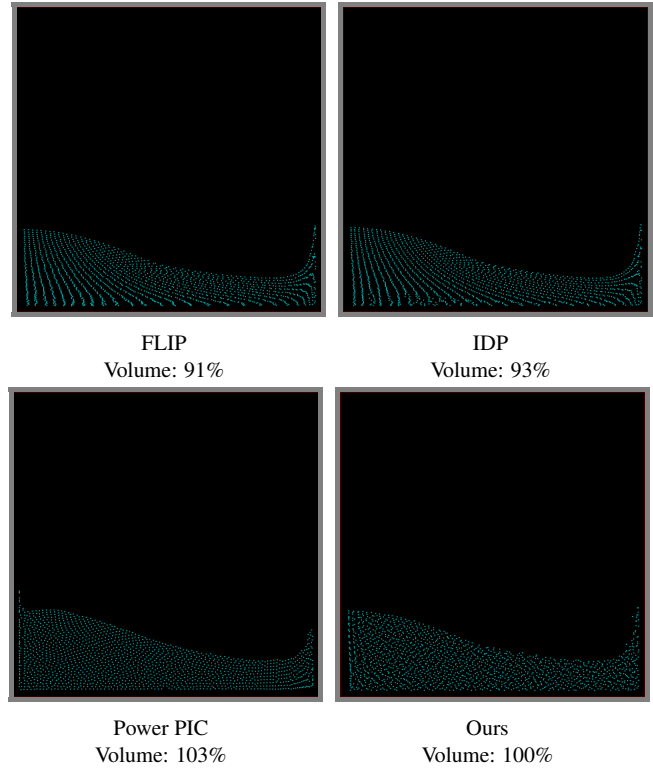

FLIP	IDP
Volume: 91%	Volume: 93%
Power PIC	Ours
Volume: 103%	Volume: 100%

Fig. 9. A frame from dam scene, when the water hits the right wall. Volume labels indicate the volume in this frame only.

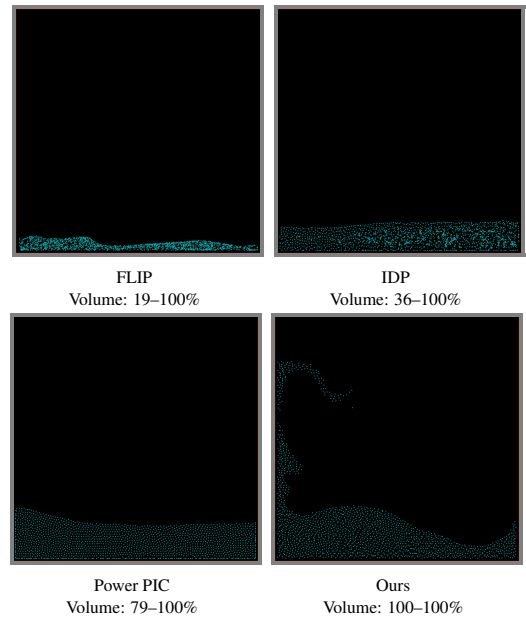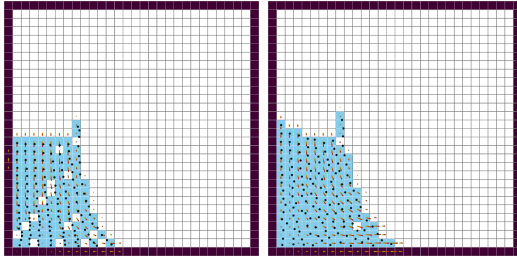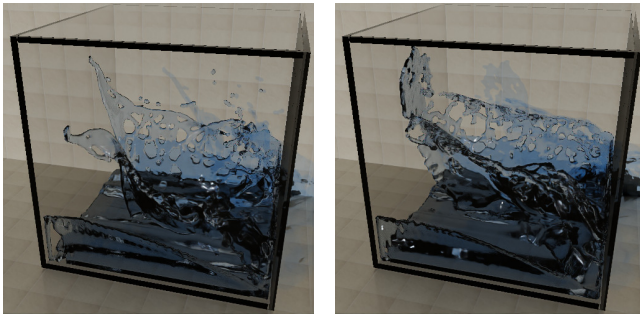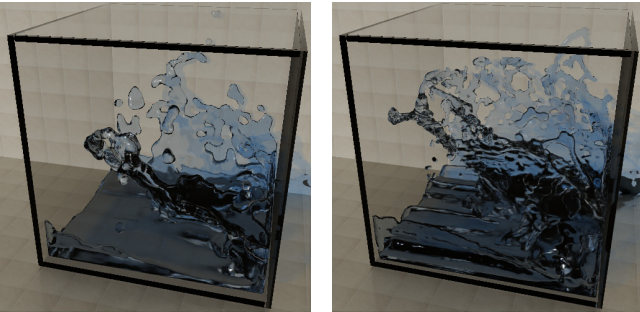|	
FLIP	IDP
Volume: 19–100%	Volume: 36–100%
Power PIC	Ours
Volume: 79–100%	Volume: 100–100%

Fig. 10. A dam scene with 1 ppc.

Fig. 11. A dam scene on a 30 × 30 grid with 1 ppc. (Left) FLIP's fluid is riddled with holes that disrupt the velocity field. (Right) our fluid has less holes, and the constraint maintains the volume.

FLIP
Volume: 65–100%

IDP
Volume: 95–100%

Our 1ppc
Volume: 100–100%

Ours
Volume: 100–108%

Fig. 12. A dam scene.

A drop of water. A drop of water is falling into a pool; see fig. 13. Notice where the splash goes. IDP's throws the splash off the center while ours keeps it centered like FLIP.

Compressing the fluid. A heavy obstacle moves at a constant velocity towards the bottom of the tank. Its movement should supersede the fluid's unless fluid speed or incompressibility are compromised. The obstacle's width is the same as the tank's, leaving no room for particles to pass it. The expected result is the obstacle moving smoothly without overlapping any particles, compressing the fluid as much as it can; see fig. 14.

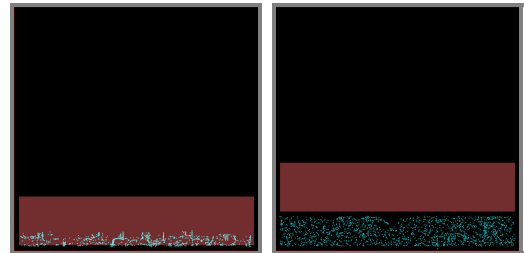We show two options for FLIP and IDP:

FLIP
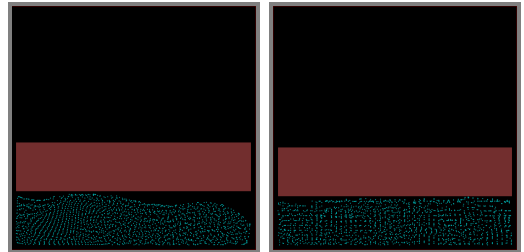Volume: 94–100%

IDP
Volume: 97–100%
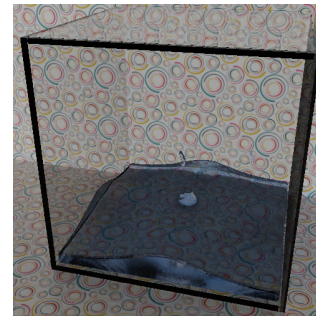
Ours
Volume: 100–100%

Fig. 13. A water drop.

FLIP, no collision detection
Volume: 0–100%

FLIP
Volume: 59–100%

IDP
Volume: 92–100%

Ours
Volume: 100–101%

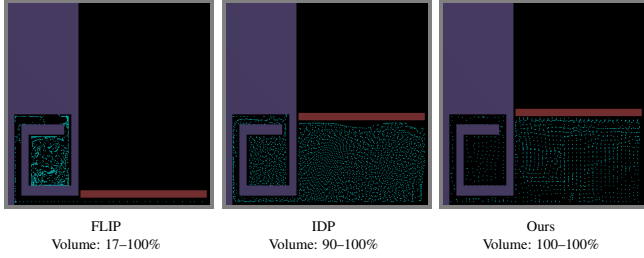Fig. 14. Compressing the fluid, the final frame.

FLIP
Volume: 17–100%

IDP
Volume: 90–100%

Ours
Volume: 100–100%

Fig. 15. A spiral, the final frame.

(1) Moving the obstacle while disregarding the fluid. Since the fluid has no room to escape, there is an inevitable overlap with the obstacle, which leads to volume loss.
(2) A naive collision detection, where the obstacle stops and waits until the fluid clears the cells that the obstacle is moving into. IDP's volume correction disperses particles, which end up in the obstacle's way and obstruct its path more than FLIP. Due to the jumpy behavior of the particles, IDP does not squeeze the fluid to the maximum possible, leaving some room for air. FLIP, on the other hand, lets the obstacle compress the fluid too much, which leads to significant volume loss. For both methods, the obstacle exhibits an undesired halting behavior.

Out method achieves the desired behavior.

A spiral. The fluid is squeezed through a narrow spiral; see fig. 15. Since the fluid's speed is limited, so is the obstacle's. We used naive collision detection for FLIP and IDP. FLIP lets the obstacle compress the fluid too much. IDP allows the obstacle to lower more than it should before it can correct the fluid, consequently losing some volume that cannot be recovered. In both methods, the obstacle's progress has more delays than necessary due to particles blocking the way. Using our method, the obstacle progresses as fast as the fluid's speed limit allows, and the fluid is compressed as much as the volume restriction allows.

A falling obstacle. An obstacle is falling into the water. FLIP and IDP behave similarly:

- Without collision detection, some particles are trapped at the bottom, which leads to volume loss.
- With collision detection, the obstacle movement is halted not long after hitting the water, far from the bottom of the tank, and there is no progress.

Using our method, the obstacle moves smoothly (like FLIP without collision detection) and there is no overlap with particles (which causes volume loss). Figure 16 shows the 2D case, and fig. 17 shows the 3D case.

Our method can move the obstacle through the fluid even without the obstacle exerting any forces on the fluid. Figure 18 shows an experiment where the boundary conditions for the pressure equation along the obstacle's boundary, eq. (12), are set to zero velocity. There is nothing to repel the fluid from the obstacle's way. As expected, when FLIP uses collision detection, the obstacle cannot penetrate the fluid. Using our method, the obstacle progresses smoothly through
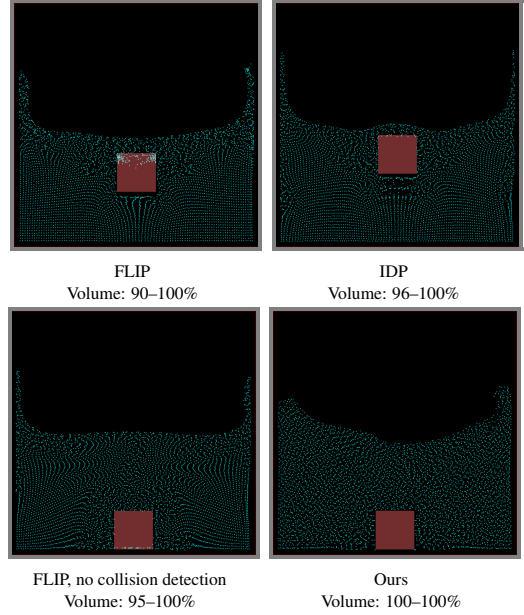

FLIP
Volume: 90–100%

IDP
Volume: 96–100%

FLIP, no collision detection
Volume: 95–100%

Ours
Volume: 100–100%

Fig. 16. A falling obstacle, the final frame.

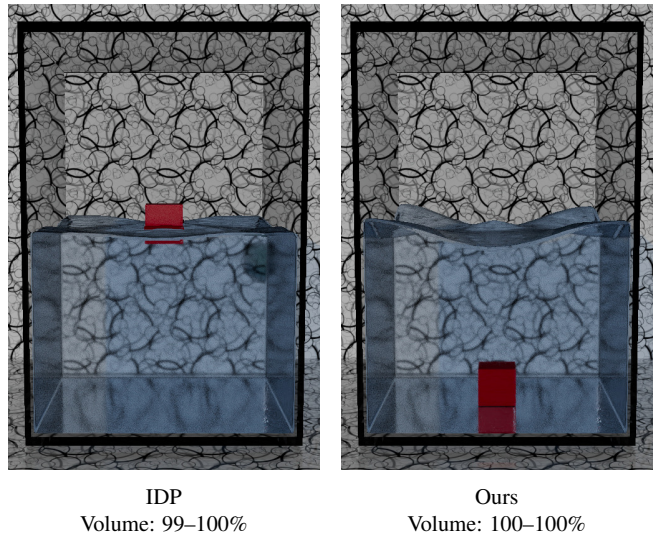
IDP
Volume: 99–100%

Ours
Volume: 100–100%

Fig. 17. A falling obstacle, the final frame. IDP with naive collision detection.

the fluid, where the correction method displaces particles out of the obstacle's way, requiring no other forces.

Figure 19 and fig. 1 show another variation with a large obstacle falling into the water. The obstacle's speed should depend on how fast the fluid can flow along the narrow paths between the obstacle and the tank. Using collision detection for FLIP and IDP, the object makes no progress after hitting the surface. Without collision detection, particles on the bottom of the tank are trapped inside the obstacle, leading to significant volume loss.
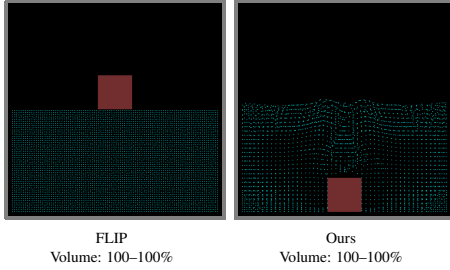
Fig. 18. A falling obstacle with boundary conditions set to zero velocity, the final frame.
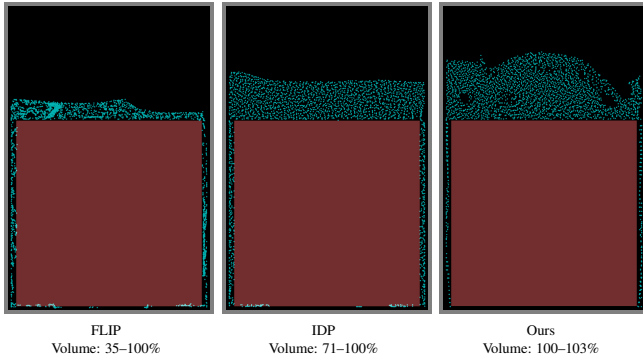

Fig. 19. A large falling obstacle, the final frame.

scene	grid	FLIP	IDP	ours
dam with emitter	50×50	49–100	92–100	100–105
dam with 1 ppc	100×100	19–100	36–100	100
compressor	50×50	59–100	92–100	100–101
spiral	50×50	17–100	90–100	100
large falling obstacle	50×75	35–100	71–100	100
dam	100×100×100	65–100	95–100	100–108
water drop	100×100×100	94–100	97–100	100
falling obstacle	100×150×100	99–100	100	100
large falling obstacle	50×75×50	48–100	79–100	100–102

Table 1. Volume preservation. A method's column shows the range of the fluid volume (presented as percentage of how much it should occupy) over all the simulation iterations, based on eq. (16).

8.2 Discussion

Methods. We focused the experiments on comparison with IDP, where its paper shows comparison with several other methods. We did not use the band method for FLIP to keep the settings close to IDP, which does not support it. Also, both methods did not have performance issues that would require it.

Behavior and volume preservation. IDP allows the fluid to violate incompressibility and increase density. In its correction step, IDP moves particles to improve density accuracy. The improvement is gradual, and the fluid may already be in a state that it cannot be recovered from. We offered several scenarios to challenge this aspect, offering two reasonable solutions to address collision detection

scene	grid	n	band	Poisson	LP	MCFP
dam	100×100×100	411K	366K	0.2	10.9	0.2
	200×200×200	3.4M	2.0M	2.5	131.7	1.3
	300×300×300	11.5M	3.6M	13.2	484.2	3.9
water drop	100×100×100	725K	228K	0.2	1.9	0.1
	200×200×200	6.2M	986K	2.3	7.8	0.7
	300×300×300	21.5M	2.3M	13.2	24	2.2
falling obstacle	100×150×100	5.7M	263K	1.2	2.8	0.2
	200×300×200	46.7M	1.0M	19.5	13.3	0.7
large falling obstacle	50×75×50	258K	183K	0.1	3.6	0.1
	100×150×100	2.2M	1.2M	1.6	73.3	1.1

Table 2. Running time. "n": number of particles in the scene. "band": the average number of particles in the band. "Poisson", "LP", and "MCFP": the average time it takes to solve a Poisson equation, the LP problem in eq. (6), and the MCFP problem in section 6.3. Average quantities are calculated over all iterations. Timings are given in seconds, rounded to one decimal place.

for FLIP and IDP: ignoring the particles and a naive detection approach. Even if the user manually selects the best of the two for each scenario, none of the behaviors are quite acceptable. The naive method caused a halting behavior or even a premature (complete) stop. Ignoring particles led to an inevitable overlap between the obstacle and the particles, which caused volume loss. Even if the loss was acceptable, the progress of the obstacle was smooth and arbitrary instead of being dependent on the fluid speed (e.g., the spiral scene). In contrast, our method strictly enforces incompressibility. A full correction is applied immediately, and the fluid cannot be compressed. Furthermore, while the obstacle's movement is prioritized, its speed is still limited by how fast the fluid can clear the way.

Our correction is mostly done by blocking and preventing particles from moving into neighboring cells rather than push them around. If a particle is moved to another cell, then the particle is positioned within the cell to be as close as possible to the location it was supposed to go. This is also prioritized over a more uniform particle distribution, like in Power PIC, which looks nice in 2D but affects the fluid behavior. IDP tends to preserve FLIP's behavior, e.g., fig. 9, where it keeps a smooth surface and clumped particles vs the other methods that add noise. However, the correction movements can also have substantial influence on the fluid's behavior, e.g., in the 3D dam when the splash overshoots and hits the left wall (fig. 12), or in the water drop scene, where the splash of water is thrown off the center (fig. 13).

Running time. In 2D, performance was not an issue for any of the methods, and a time step took less than a second. Table 2 gives timings for 3D scenes with varying grid sizes.

The timing of solving a Poisson equation depends on the number of fluid cells. For methods that preserve incompressibility, the number of fluid cells is approximately the number of particles divided by μ.

MCFP's timing depends on the number of fluid cells. In a typical scene, alg. 3 needs to be executed rarely more than once in a time step if at all. However, in a scene such as the large falling object, there are time steps where a few calls are needed. For example,
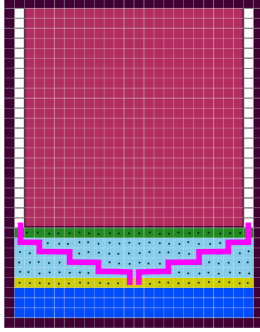
Fig. 20. An extreme (hypothetical) case of paths (in magenta) that start in the middle of the band, creating a V shape and blocking other paths.

consider the extreme scenario in fig. 20. Two low-cost paths start in two incident cells in the middle of the band interface and lead to the surface along the narrow passages between the obstacle and the tank, creating a V shape. Since the algorithm finds non-overlapping paths, these two paths block all other paths, and another call to alg. 3 is necessary if more paths are needed. This increases the MCFP's average time for such a scene.

The time required to solve the LP can vary, depending on how hard the problem is, which does not necessarily depend on the number of particles and grid size. For example, in the first iterations of the water drop scene, before the drop hits the pool, the LP solution is close to the ideal particle positions, and the LP is solved in 7 seconds for a 300^3 grid. On the other hand, in the large obstacle scene, the objective of the LP includes the obstacle, and the solution decides if the obstacle moves or not. This creates a dependency between cells in $C_{\text{new\_solid}}$ that are fluid or incident to fluid cells, which is similar to a global dense constraint. A solution to the LP in this case can take a few minutes even for a $50 \times 75 \times 50$ grid.

An obvious advantage of using the band method is the reduction in the number of particles, e.g., from 46.7 million to an average of one million in the falling obstacle scene with the $200 \times 300 \times 200$ grid. But another advantage of the band method is that it may accelerate the LP even if the number of particles is not reduced significantly. For example, in the 100^3 dam scene, when not using the band method, the average LP time is 38.9 seconds. When using the band method, the number of particles is reduced only by 16%, but the average LP time is reduced to 8.6 seconds. This is because the band method simplifies the problem. Intuitively, the constraints become local, where particles can simply follow gravity into the deep with no restriction (which is corrected by the MCFP in the second step). This is not the case without the band method, where the flow reaches the bottom of the tank or abides the volume constraint, and it needs to go side-ways and up, which is a more global behavior.

Comparing the three methods, FLIP is the fastest since it requires only one Poisson solution, which the two other methods require as well. IDP requires solving an additional Poisson equation while our method needs to solve an LP and an MCFP. The cost of the additional Poisson solution is reasonable, and more importantly it is predictable since it depends on the number of fluid cells. Our method mostly runs in a reasonable time on moderate size grids. But in some

cases, while solving the MCFP remains reasonably fast, solving the LP can take a few minutes, which leaves room for improvement.

9 CONCLUSION

We proposed a method that constrains particles to grid cells to enforce our definition of discrete incompressibility. While the fluid can still inflate with air bubbles, we show experimentally that the expansion is moderate. Keeping strict incompressibility is one advantage over previous work, which instead gradually corrects the fluid over time. One issue with gradual correction is that volume preservation is not perfect, which may cause noticeable artifacts. A more sever issue is that the fluid can reach a state that is irrecoverable.

Our framework can be further exploited in other applications, and we show examples of coupling with solids which naive solutions applied to the state of the art fail to handle adequately.

The main drawback of the method is performance. In each iteration, an LP is solved. Besides the number of particles, the fluid configuration affects the running time, which may be longer than desired. We offered acceleration via an adapted version of the band method that enforces incompressibility, and we showed experimentally that it performs reasonably on moderate size grids. The fastest variation of our band method solves an easier LP, followed by an additional correction that solves an MCFP. While the solution is not optimal, the result is reasonable for the affected amount of particles. A future avenue could be to find faster alternatives to the LP.

REFERENCES

Ryoichi Ando, Nils Thuerey, and Chris Wojtan. 2015. "A stream function solver for liquid simulations." *ACM Transactions on Graphics*, 34, 4, 1–9.

Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. 2012. "Preserving fluid sheets with adaptively sampled anisotropic particles." *IEEE Transactions on Visualization and Computer Graphics*, 18, 8, 1202–1214.

Ryoichi Ando, Nils Thürey, and Chris Wojtan. 2013. "Highly adaptive liquid simulations on tetrahedral meshes." *ACM Transactions on Graphics (TOG)*, 32, 4, 1–10.

Stefan Band, Christoph Gissler, Markus Ihmsen, Jens Cornelis, Andreas Peer, and Matthias Teschner. 2018. "Pressure boundaries for implicit incompressible SPH." *ACM Transactions on Graphics*, 37, 2, 1–11.

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

Jan Bender and Dan Koschier. 2017. "Divergence-Free SPH for Incompressible and Viscous Fluids." *IEEE Transactions on Visualization and Computer Graphics*, 23, 3, 1193–1206.

Robert Bridson. 2015. *Fluid simulation for computer graphics*. CRC press.

Fernando De Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. "Power particles: an incompressible fluid solver based on power diagrams." *ACM Trans. Graph.*, 34, 4, 50–1.

Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. "Fluid simulation using laplacian eigenfunctions." *ACM Transactions on Graphics*, 31, 1, 1–11.

Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. 2007. "Stable, circulation-preserving, simplicial fluids." *ACM Transactions on Graphics (TOG)*, 26, 1.

Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. 2016. "Narrow band FLIP for liquid simulations." In: *Computer Graphics Forum* 2. Vol. 35, 225–232.

Nick Foster and Dimitri Metaxas. 1996. "Realistic animation of liquids." *Graphical models and image processing*, 58, 5, 471–483.

Chuyuan Fu, Qi Guo, Theodore Gast, Chenfanfu Jiang, and Joseph Teran. 2017. "A polynomial particle-in-cell method." *ACM Transactions on Graphics*, 36, 6, 1–12.

Dan Gerszewski and Adam W Bargteil. 2013. "Physics-based animation of large-scale splashing liquids." *ACM Trans. Graph.*, 32, 6, 185–1.

Gurobi. 2018. *Gurobi Optimizer Reference Manual*. (2018). http://www.gurobi.com.

Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. 2014. "Implicit Incompressible SPH." *IEEE Transactions on Visualization and Computer Graphics*, 20, 3, 426–435.
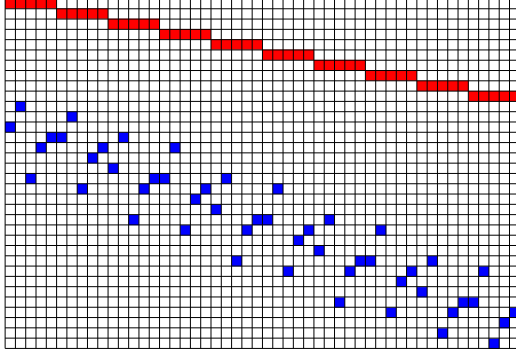
Fig. 21. Visualizing a TU matrix. Zeros are in white, the rest are ones. The rows with the red cells correspond to eq. (6c). The rows with the blue cells correspond to eqs. (6d) to (6f) (without duplicate rows). In both the red and blue set of rows, each column sums up to one.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. "The affine particle-in-cell method." *ACM Transactions on Graphics*, 34, 4, 1–10.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. 2016. "The material point method for simulating continuum materials." In: *Acm siggraph 2016 courses*, 1–52.

Byungmoon Kim, Yingjie Liu, Ignacio Llamas, Xiangmin Jiao, and Jarek Rossignac. 2007. "Simulation of bubbles in foam with the volume control method." *ACM Transactions on Graphics*, 26, 3, 98–es.

Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. 2022. "A Survey on SPH Methods in Computer Graphics." *Computer Graphics Forum*, 41, 2.

Tassilo Kugelstadt, Andreas Longva, Nils Thuerey, and Jan Bender. 2019. "Implicit density projection for volume conserving liquids." *IEEE Transactions on Visualization and Computer Graphics*, 27, 4, 2385–2395.

Miles Macklin and Matthias Müller. 2013. "Position based fluids." *ACM Transactions on Graphics*, 32, 4, 1–12.

Matthias Müller, David Charypar, and Markus H Gross. 2003. "Particle-based fluid simulation for interactive applications." In: *Symposium on Computer animation*, 154–159.

Rahul Narain, Abhinav Golas, and Ming C. Lin. 2010. "Free-Flowing Granular Materials with Two-Way Solid Coupling." *ACM Transactions on Graphics*, 29, 6, 1–10.

Ziyin Qu, Minchen Li, Fernando De Goes, and Chenfanfu Jiang. 2022. "The power particle-in-cell method." *ACM Transactions on Graphics*, 41, 4.

Craig W Reynolds. 1987. "Flocks, herds and schools: A distributed behavioral model." In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 25–34.

Takahiro Sato, Christopher Wojtan, Nils Thuerey, Takeo Igarashi, and Ryoichi Ando. 2018. "Extended narrow band FLIP for liquid simulations." In: *Computer Graphics Forum* 2. Vol. 37, 169–177.

Alexander Schrijver. 1998. *Theory of linear and integer programming*. John Wiley & Sons.

Jos Stam. 1999. "Stable fluids." In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 121–128.

Tetsuya Takahashi and Ming C Lin. 2019. "A Geometrically Consistent Viscous Fluid Solver with Two-Way Fluid-Solid Coupling." In: *Computer Graphics Forum* 2. Vol. 38, 49–58.

Nils Thuerey and Tobias Pfaff. 2018. *MantaFlow. http://mantaflow.com.* (2018).

Kiwon Um, Seungho Baek, and JungHyun Han. 2014. "Advanced hybrid particle-grid method with sub-grid particle correction." In: *Computer Graphics Forum* 7. Vol. 33, 209–218.

Yongning Zhu and Robert Bridson. 2005. "Animating sand as a fluid." *ACM Transactions on Graphics*, 24, 3, 965–972.

A PROOFS

PROPOSITION 5.1. *The LP relaxation of the ILP in eq. (6), which uses continuous variables, has the same optimal solution.*

PROOF. Transform the LP in eq. (6) into a canonical form $\max\{cy \mid Ay \leq d\}$, where A is a matrix, and c, y, and d are vectors:

- Change the objective to max by negating it.
- Replace an equality constraint with two inequalities (bounding the LHS expression from both sides).
- Change $\geq$ inequalities to $\leq$ by negating them.
- Convert the problem into a matrix form. An expression that is bounded from both sides (which appears in two inequalities, e.g., eq. (6e) or a transformed equality constraint) appears as two identical rows in A up to a sign.

The feasible region of an LP is a polyhedron (an intersection of hyperplanes). Due to linearity, an optimal solution (an extreme point) is at a polyhedron vertex. The polyhedron has vertices with integral coordinates if the matrix A is totally unimodular (TU) and d (the RHS) is integral [Schrijver 1998, theorem 19.1]. A matrix is TU if each of its subdeterminants is $\in \{0, \pm 1\}$.

LEMMA A.1. *A is TU.*

Proof by induction on the size $k \times k$ of a square submatrix of A.
Base case: holds for $k = 1$ since each entry of A is $\in \{0, \pm 1\}$.
Induction step: Assume the determinant of a $k \times k$ submatrix of A is $\in \{0, \pm 1\}$, prove for a submatrix $B \in \mathbb{R}^{(k+1) \times (k+1)}$. Possible cases:

- B has a row or column of zeros. Then, it is rank-deficient, and its determinant is zero. Similarly if B has a duplicate row up to a sign (e.g., two inequalities that bound the same expression, and both rows are in B).
- B has a row with a single nonzero B_{ij} (e.g., eq. (6b)). Then, consider the Laplace expansion along this row. It will be equal to B_{ij} times a $k \times k$ cofactor of B, which according to the assumption is $\in \{0, \pm 1\}$. Similarly if B has a column with a single nonzero
- Each variable corresponds to a particle movement, which ends up in a specific cell. Therefore, each variable appears only once in eqs. (6d) to (6f). Moreover, each variable appears once in eq. (6c). This leaves us with the last case where each column of B has two nonzeros. The nonzeros in each row are either all 1 or -1. Multiply each negative row by -1, which may only affect the sign of the determinant. Divide B into two matrices

$$\begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \quad B_1 \in \mathbb{R}^{l \times (k+1)}, \quad B_2 \in \mathbb{R}^{(k-l+1) \times (k+1)}$$

such that a column in each matrix has a single 1; see fig. 21 for illustration. Let $v \in \mathbb{R}^k$ be the vector

$$v := \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}, \quad v_1 := \mathbb{1} \in \mathbb{R}^l, \quad v_2 := -\mathbb{1} \in \mathbb{R}^{k-l+1},$$

where $\mathbb{1}$ is a vector of ones. v is in the null space of $B^\top$, and thus $\det B = 0$.

□