Contents

Cell-Constrained Particles for Incompressible Fluids

Zohar Levi[a]

[a] *An independent researcher, , , Wellington, New Zealand*

Abstract

Incompressibility is a fundamental condition in most fluid models. Accumulation of simulation errors violates it and causes fluid volume loss. Prior work has proposed correction methods to combat this drift, but they remain approximate and can fail in extreme scenarios. We present a particle-in-cell method that *strictly* enforces a grid-based definition of discrete incompressibility at every time step.

We formulate a linear programming (LP) problem that bounds the number of particles that end up in each grid cell. To scale this to large 3D domains, we introduce a narrow-band variant with specialized band-interface constraints to ensure volume preservation. Further acceleration is achieved by simplifying the problem and adding a band-specific correction step that is formulated as a minimum-cost flow problem (MCFP).

We also address coupling with moving solids by incorporating obstacle-aware penalties directly into our optimization. In extreme test scenes, we demonstrate strict volume preservation and robust behavior where state-of-the-art methods exhibit noticeable volume drift or artifacts.

Keywords: fluid simulation, incompressibility, discrete incompressibility, linear programming, minimum-cost flow

1. Introduction

Fluids exhibit rich, complex behaviors, and faithfully simulating them has been an active area of research in computer graphics to improve the realism of animations of a wide 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 the fluid during a simulation. Specifically, the PDEs in eq. (1) describe the flow of the fluid, in terms of velocity, as an evolution of a system during an infinitesimal time step. The movement of the fluid during the time step is restricted to be divergence-free. However, inaccuracies and numerical errors accumulate over time and become pronounced. Nothing accounts for the total change over time in the fluid's volume from its initial state because the preservation applies only to the immediate change that occurs over a time step.

To limit this drift, prior work has introduced correction methods ranging from improving particle spacing [1] and enforcing a stricter density correction via an additional solution of a Poisson equation [15] to treating particles as volume parcels with prescribed volume [25]. Despite their strengths, these correction methods remain imperfect—subject to numerical inaccuracies—and can fail in challenging scenarios. The common approach is to gradually—over a few time steps—remedy volume error by adjusting particle positions. Until this process converges—assuming no new errors arise—the fluid in its current state suffers from volume loss, often producing visible artifacts. Worse, the simulation can enter an unrecoverable state, where volume error cannot be rectified. We illustrate these drawbacks in a few scenes in section 8.1.

Our approach builds on the PIC/FLIP hybrid framework [4], which represents fluids with both an Eulerian grid and Lagrangian particles. We enforce volume preservation by defining a grid-based discrete incompressibility: particles are confined to cells, and each cell's particle count is strictly bounded at every time step.

To do so, we first tackle how to measure volume and cast incompressibility across these two representations. In section 4, we define discrete incompressibility via per-cell particle occupancy. We then insert a correction pass into the FLIP pipeline (section 3) that restores this property at each time step. By restricting particle movements to a fixed set of neighboring cells (section 5), we recast the correction as an integer linear programming (ILP) problem with hard constraints on per-cell occupancy, guaranteeing exact incompressibility.

To improve running time, we show that the ILP can be relaxed to a linear programming (LP) problem. This results in an optimization problem with smooth variables that still enforces integer constraints on the number of particles per cell. Nevertheless, this formulation does not scale well, and running time can be an issue even for moderate-sized 3D grids. We offer a variant of the (narrow) band method [8], tailored to our approach to preserving incompressibility (section 6). This approach requires monitoring the number of particles that go into and out of the band region. This is achieved via the formulation of an additional band-specific constraint (section 6.1) that

FLIP
Volume: 48–100%

IDP
Volume: 79–100%

Ours
Volume: 100–102%

Figure 1: A large box is dropped into water. For FLIP [36] and IDP [15], particles at the bottom of the tank do not manage to clear the path quickly enough, and they become trapped inside the solid box, causing significant volume loss. The volume range over all time steps is indicated below each image.

restricts particle movements near the band interface.

To further accelerate the method, we propose two faster variants that restrict particle movements into and out of the band. Both variants first solve the LP as is—without the additional band constraint—producing an easier problem that is faster to solve. This is followed by a second specialized step to correct the band interface and deeper parts of the fluid. The first variant solves the LP a second time but with a shorter one-way band constraint (section 6.2), reducing running time.

The second variant (section 6.3) models the fluid as a flow in a discrete graph. Grid cells are treated as graph vertices, and particle movements are treated as graph edges. Based on this graph representation, we formulate a minimum-cost flow problem (MCFP) and employ a Dijkstra-based solver (section 6.3). Overall, while solving the LP does not scale well, this fastest variant of the band method achieves practical performance on moderate-sized 3D grids (table 2).

Coupling with solids is a fundamental challenge, and we address it within our incompressibility-preserving framework (section 7). In our evaluation, we devised extreme scenarios (section 8.1) that highlight our method's advantage over the state of the art. One aspect that we show is that gradual correction over time, a property that prior works have in common, may not be adequate in certain scenarios: There may not be an opportunity to correct the fluid after an obstacle moves. By enforcing incompressibility at each time step, our approach remains robust even in these cases.

2. Related Work

For background on grid-based fluid simulation, see [4]. We will focus our review of prior work on methods that are related to volume preservation.

Kim et al. [13] simulate bubbles using the level-set method. They track volume changes of each connected region and compensate for errors via divergence control. Lentine et al. [16] augment the level-set function with an advected color function to improve volume preservation.

Smoothed-particle hydrodynamics (SPH) [14] takes a purely Lagrangian view, representing fluids with particles. Simple, intuitive methods such as [21]—reminiscent of the boids algorithm [26]—apply local rules to each particle based on its neighborhood, resulting in emergent global behavior. Macklin and Müller [20] embed an iterative density correction within position-based dynamics [22], enforcing incompressibility via per-particle density constraints. The method remains stable at large time steps and adds an artificial pressure term to prevent clustering. Bender and Koschier [3] combine two pressure solvers, one enforcing a divergence-free velocity field and the other a constant-density condition. Band et al. [2] improve implicit incompressible SPH (IISPH) [11] by enforcing a consistent pressure gradient at boundary samples through an alternative discretization of the pressure equation. In a hybrid SPH and particle level set method, Losasso et al. [19] proposed a local volume correction that adds a mass-conservation-derived term to the Poisson equation, discouraging overly dense particle clusters in localized fluid volumes. Takahashi and Lin [32] simulate viscous materials with strong

two-way coupling with solids. They apply position correction based on density constraints.

Hybrid schemes based on the particle-in-cell (PIC) and fluid-implicit-particle (FLIP) methods [36] use a dual view combining a grid and particles for fluid representation. This approach parallels the material point method (MPM) [12], which was used to simulate a larger variety of materials, including those described by elasto-plastic constitutive models. Ando, Thurey, and Tsuruno [1] detect and preserve thin fluid sheets, which are reconstructed using anisotropic kernels. An anisotropic position correction is performed for even particle distribution. Um, Baek, and Han [34] apply sub-grid particle correction for improved particle distribution. The band method [8] keeps particles only within a narrow band of the liquid surface for performance. Sato et al. [28] extend the band method with particle correction based on [1] to better distribute particles near the surface.

From the mass conservation law, Kugelstadt et al. [15] (IDP) derive a pressure Poisson equation which takes density deviation into account. They add a density correction step—solving an extra Poisson equation—to recover fluid volume. Density correction was previously performed using the so-called unilateral incompressibility constraint, which was applied to free-flowing granular materials [23] and splashing liquids [9].

The power particles method [7] treats particles as fixed-volume sites and partitions the fluid domain via a power diagram. Prescribed particle volumes enable precise fluid-volume control and improve distribution. Power PIC [25] boosts performance by recasting [7] as a transportation problem solved efficiently with Sinkhorn's iterative algorithm. In two-phase simulations, an estimated surface provides an air-occupancy baseline for slack-air variables, which fill gaps between prescribed particle volume and cell volume. Lévy [17] also employ power diagrams with transport formulations. Zhai et al. [35] implement power particles on the GPU. Li et al. [18] use particle trajectories to establish flow maps and tailor path integrals of physical quantities to reformulate the Poisson problem, sharing implementation similarities with the power particles method.

Enhancing simulation accuracy mitigates the problem and assists in volume preservation by reducing errors that cause volume variation up front. For example, improving the velocity field interpolation can reduce errors [5, 30, 27]. Qiu, Yu, and Fedkiw [24] add pressure degrees of freedom onto surfaces of rigid bodies to improve solid–fluid coupling in thin gaps that are smaller than the grid resolution. Although improving simulation accuracy reduces the need for correction methods in some scenes, discretization errors can still accumulate significantly in others, necessitating correction.

Related lattice-based approaches such as the Lattice-Boltzmann Method (LBM) [6] also propagate (virtual) particles along fixed neighbor links. However, LBM evolves mesoscopic distribution functions via local streaming and collision rules to recover continuum fluid behavior, whereas our method performs a deterministic global optimization over discrete particle movements to enforce incompressibility and placement constraints (see section 5).

3. Background

We model a fluid on a domain $\Omega \subset \mathbb{R}^d$, $d \in \{2, 3\}$, using the Euler equations for inviscid, incompressible flows:

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

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

where $u, f \in \mathbb{R}^d$ and $p, \rho \in \mathbb{R}$ denote velocity, external forces, pressure, and density. $\frac{\mathsf{D}}{\mathsf{D}\, t}$ denotes the material derivative. Equation (1b) enforces a divergence-free velocity field, ensuring incompressibility.

The FLIP method is a hybrid discretization method that combines the Eulerian and Lagrangian views [4]. The domain is discretized with a regular (square or cubic) grid, and the fluid is represented by particles. The FLIP algorithm alternates between the views, solving for pressure on the grid and advecting quantities via particles. Discretization inaccuracies accumulate over time, causing visible volume change; a correction step is therefore required to conserve volume—ideally keeping the fluid's volume constant throughout the simulation. The algorithm is listed in alg. 1 for a single time step. The optional correction step is omitted from the original FLIP method; IDP and our method implement it differently.

Algorithm 1: One time step of the FLIP method

1. Transfer velocity from particles to grid `// particles are at` $\bar{x}$
2. Apply external forces to grid
3. Project velocity onto divergence-free fields `// solve for pressure`
4. Transfer velocity from grid to particles
5. Advect particles `// particles are at` $\hat{x}$
6. Correct particle positions `// particles are at` x

4. Discrete Incompressibility

We propose a definition for discrete incompressibility based on particle occupancy within the grid. 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 for a smoother fluid flow: Each fluid cell keeps a constant number $\mu \in \mathbb{Z}$ of particles throughout the simulation. μ is given, and it is usually based on the initial fluid state. Typically, we use $\mu = 4$ in 2D and $\mu = 8$ in 3D.

Preserving this condition is performed in the correction step described below, starting with notation. Let $\mathcal{C}$ be the set of grid cells that cover Ω. We associate markings with grid cells; the markings describe their characteristics. Each marking has a subset of cells that are marked with it. Initially, we use the disjoint subsets $\mathcal{C}_{\text{empty}}$, $\mathcal{C}_{\text{solid}}$, and $\mathcal{C}_{\text{fluid}}$ 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:

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

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

$$|\gamma_c| \leq 0 , \quad \forall c \in \mathcal{C}_{\text{solid}} , \tag{2c}$$

where $x_j, \hat{x}_j \in \Omega$ are the new and ideal positions of the j-th particle, and $|\gamma_c|$ denotes the number of particles in cell c. The sets γ are determined by x (which we are solving for), i.e., if x_j is within cell c, then γ_c will contain the j-th particle.

Cell markings $\mathcal{C}_{\text{empty}}$ and $\mathcal{C}_{\text{fluid}}$ are also determined by x. On the other hand, cell markings $\mathcal{C}_{\text{solid}}$ are determined by objects in the scene (set by the user), and eq. (2c) ensures that those marked cells do not contain particles. $\mathcal{C}_{\text{empty}}$, by contrast, does not require a constraint, since it is determined by particle positions. That is, solids in the scene are set by the user, and particles must avoid them. The rest of the cells either do or do not contain particles and are marked as fluid or empty accordingly. Any cell $c \in \mathcal{C}_{\text{empty}}$ automatically satisfies the incompressibility constraints because its particle count is zero: $|\gamma_c| = 0$. Therefore, no additional constraint is imposed on cells in $\mathcal{C}_{\text{empty}}$.

While the problem in eq. (2) clearly expresses what we want, it is still unclear how to implement it. There is the challenge of connecting particle positions x, which we are solving for, to the sets γ, which relate a particle to the cell that contains it. The constraints eq. (2b) and eq. (2c) are implicitly conditional—they depend on whether the j-th particle ends up in cell c—and it is still unclear how to write them as a canonical optimization problem. This will be addressed in section 5, but first we will improve the model for smoother fluid movement.

4.1. Surface and Bubbles

The discrete incompressibility condition in the previous section is too restrictive, and it forces the fluid to propagate rigidly. We will relax this condition in a reasonable way to enable smooth movement of the fluid by allowing a particle on the fringe of the fluid to explore by itself a neighboring empty cell.

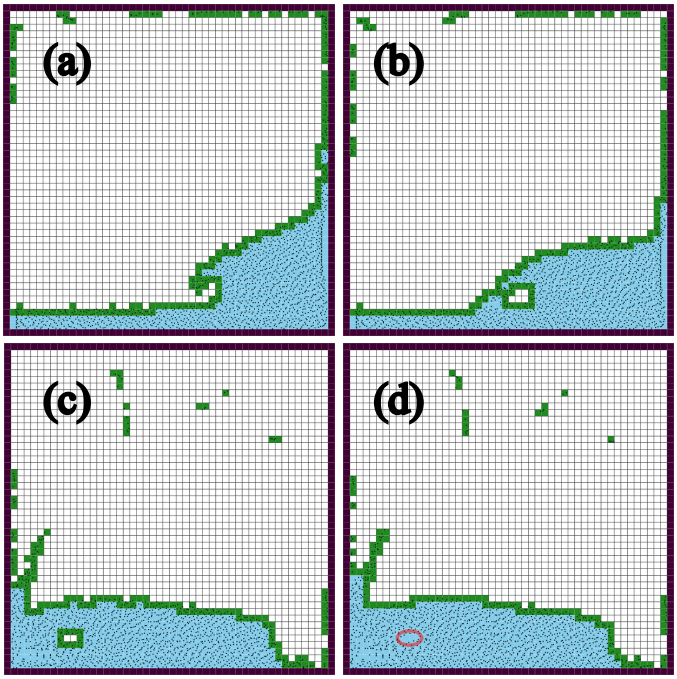


Figure 2: The surface (green cells) of a breaking wave (a) closes upon itself, creating an air pocket (b). The air pocket shrinks (c) until surface cells no longer have empty neighbors and become inner cells (d). These former surface cells may have fewer than μ particles, and in such a case we say that they contain air bubbles (an example is circled in red).

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 6-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 1 (surface)**.** *Given cell markings, we define a fluid cell as surface if it neighbors a cell in a Moore neighborhood that is not a fluid cell and is not on the domain boundary. We partition $\mathcal{C}_{fluid}$ into surface cells $\mathcal{C}_{surface}$ and inner cells $\mathcal{C}_{inner}$.*

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). These solid cells are marked by the user, who sets the scene.

We start with the 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 mean 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. Instead, it is desirable that the fluid be able to propagate smoothly at any speed, which involves a particle moving alone into a neighboring empty cell (without dragging $\mu-1$ particles with it). To allow shape flexibility and individual particle movement between cells, we permit surface cells to have fewer than μ 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 to solve.

Definition 2 (bubble)**.** *An inner cell $c \in \mathcal{C}_{inner}$ 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 when a surface cell suddenly becomes an inner cell since it no longer has any incident empty cells. A surface cell $c \in \mathcal{C}_{\text{surface}}$ that transitions into $\mathcal{C}_{\text{inner}}$ may contain fewer than μ particles; equivalently, $|\gamma_c| \leq \mu$. Such partial occupancy can occur naturally during fluid evolution, and we accept these cases rather than enforcing global corrections that would otherwise restore μ particles to every inner cell.

Following that, we allow inner fluid cells to keep (but not grow) air bubbles. Specifically, instead of requiring each inner cell $c \in \mathcal{C}_{\text{inner}}$ to contain at least μ ppc, we require that an inner cell c does not lose particles and has at least as many particles as in the previous time step. That is, the particle count $|\gamma_c|$ does not decrease after c becomes an inner cell. Formally, for every $c \in \mathcal{C}_{\text{inner}}$ we enforce $|\bar{\gamma}_c| \leq |\gamma_c|$, where $\bar{\gamma}_c$ denotes the particle count in cell c at the previous time step. Note that compared to a surface cell, this is still a restriction since we do not want the fluid to expand arbitrarily. See fig. 2 for an illustration.

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

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

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

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

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

In this formulation, inner cells can have fewer than μ particles (air bubbles), but they cannot lose particles. Surface cells are given more flexibility, and they can lose particles. Note that while the relaxed constraints allow reasonable fluid 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 no particle moves 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, we define the set $\mathcal{D}$ that consists of the columns of the $d \times d$ identity matrix and their negations, along with the zero vector that signifies staying in the same cell. For example, in 2D:

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

Let $m \coloneqq |\mathcal{D}|$ (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 will choose one direction from $\mathcal{D}$ by solving the problem in eq. (7). Let $b \in \mathbb{Z}_2^{m \times n}$ be an $m \times n$ binary matrix that chooses a direction for each particle. The j-th column is assigned to the j-th particle, and it contains a single nonzero in the entry that corresponds to the particle's chosen direction.

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

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

where $v \coloneqq q - \phi_{\text{center}}(c)$, a subscript k denotes the k-th component of a vector (or coordinate of a point) in Ω, and ϵ (=0.01) is a small margin based on the grid size (we used an integer grid).

Let $\bar{x} \in \Omega^n$ be the particle positions at the end of the previous time step. The matrix entry b_{ij} corresponds to a possible grid movement $\mathcal{D}_i$ for the j-th 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 with respect to its ideal position and σ_{obj}:

$$\xi_{ij} \coloneqq \phi_{\text{close}}\left(\hat{x}_j, \mathit{cell}\left(\bar{x}_j + \mathcal{D}_i\right)\right) \ , \tag{6}$$

where $\bar{x}_j$ denotes the position of the j-th particle in the previous time step, and $\text{cell}(\cdot)$ denotes 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 c. Each index pair is

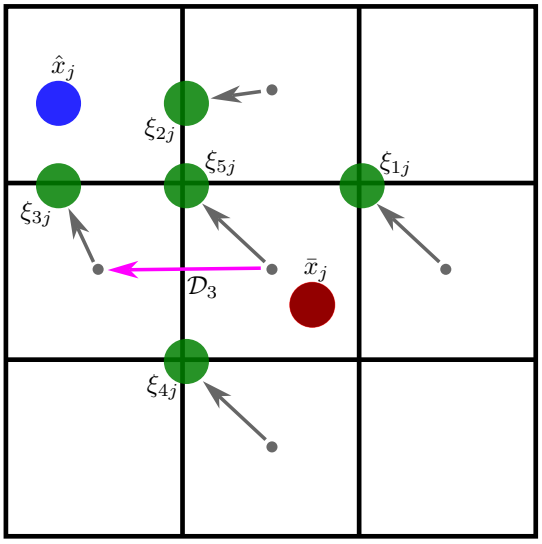


Figure 3: Possible positions ξ_{ij} (in green) of the j-th particle. A gray arrow points from the center of the cell that the particle is confined to to the closest location in that cell (up to a margin ϵ) to $\hat{x}_j$ (in blue). If $\hat{x}_j$ was in one of the five possible cells, then the corresponding ξ_{ij} to that cell would have coincided with it. All the possible positions ξ_{ij} are known, and a solution to eq. (7) selects one as the position x_j of the particle at the end of the time step. There are five cells corresponding to $\mathcal{D}$. One example of a discrete direction is given 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 $\mathcal{D}_3$.

associated with a particle movement option and consists of a direction index and a particle index:

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

The problem becomes:

$$\min_{b} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}\sigma_{\text{obj}}\left(\xi_{ij}, \hat{x}_j\right) \tag{7a}$$

$$\text{s.t.} \quad 0 \le b_{ij} \le 1\ , \quad \forall b_{ij} \in b \tag{7b}$$

$$\sum_{i=1}^{m} b_{ij} = 1\ , \quad j = 1 \dots n \tag{7c}$$

$$\sum_{(i,j)\in\tilde{\gamma}_c} b_{ij} \le \mu\ , \quad \forall c \in \bar{\mathcal{C}}_{\text{empty}} \cup \bar{\mathcal{C}}_{\text{surface}} \tag{7d}$$

$$|\bar{\gamma}_c| \le \sum_{(i,j)\in\tilde{\gamma}_c} b_{ij} \le \mu\ , \quad \forall c \in \bar{\mathcal{C}}_{\text{inner}} \tag{7e}$$

$$\sum_{(i,j)\in\tilde{\gamma}_c} b_{ij} \le 0\ , \quad \forall c \in \bar{\mathcal{C}}_{\text{solid}}\ . \tag{7f}$$

Details:

- The objective in eq. (7a) 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 (7b) asserts the range of binary variables.
- Equation (7c) forces a single selected direction for each particle.
- Equation (7d) and eq. (7e) 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. $\bar{\mathcal{C}}$ refers to the markings in the previous time step.
- Equation (7f) is similar to eq. (3d)

See fig. 4 for an example.

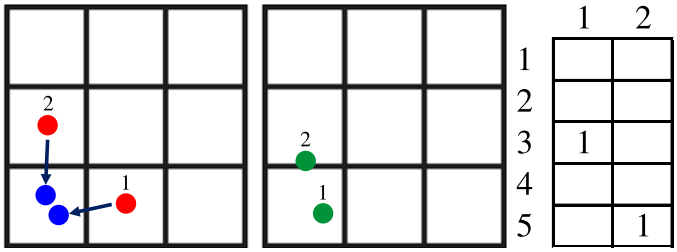


Figure 4: A 3×3 grid with two particles. The particle positions from the previous step ($\bar{x}_j$) are in red. The ideal particle positions after advection ($\hat{x}_j$) are in blue. The discrete incompressibility constraint is $\mu = 1$ ppc. The optimal particle positions within the cells associated with a selection from possible grid movements (ξ_{ij}) are in green. This selection corresponds to the matrix b on the right, which is a minimizer of eq. (7). The 1st particle moves to the neighboring cell to its left, and it reaches its ideal position. There is no added cost for this particle to the objective function. The 2nd particle stays in its current cell, but it gets as close as it can to its ideal position. The added cost to the objective function is the squared distance between its final (selected) position and its ideal position.

In eq. (7d), we use the surface marking from the previous time step 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 time step, which may belong to the layer of inner cells incident to the surface in this time step), 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, i.e., the i-th entry, the single nonzero in the j-th column of the solution b^*.

The problem in eq. (7) 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). This 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 which is generally faster to solve.

Proposition 1. *The LP relaxation of the ILP in eq. (7), which uses continuous variables, has the same optimal solution.*

See the proof in appendix Appendix A.

To summarize the correction steps:

- Based on the previous particle positions $\bar{x}$, determine the cell markings $\bar{\mathcal{C}}$ and the sets $\bar{\gamma}$.
- Calculate the optimal positions ξ from eq. (6) and costs σ_{obj}.
- Solve the LP in eq. (7) for b.
- Determine new positions x based on the solution $b^\star$.

6. The Band Method

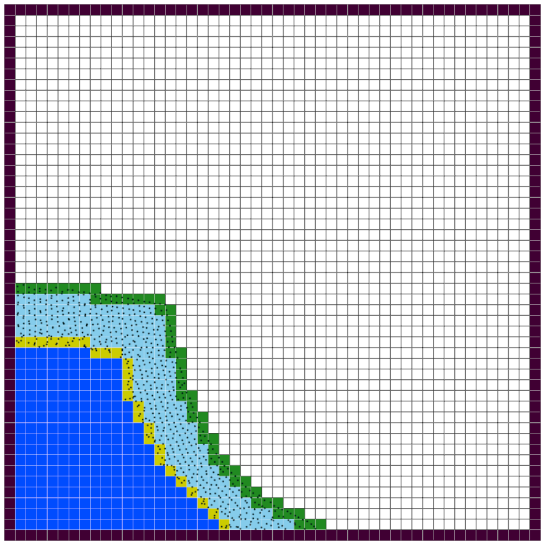


Figure 5: A band. Deep cells ($\mathcal{C}_{<-R}$) in dark blue, band interface ($\mathcal{C}_{-R}$) in yellow, surface ($\mathcal{C}_0$) in green, and the rest of the band ($\mathcal{C}_{-R<\beta<0}$) in light blue.

Solving the LP in eq. (7) does not scale well, and for large 3D grids, we propose an incompressible variant of the band method [8]. 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 the 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 time step 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 [8] uses an approximate distance from the surface to define the band, our discrete approach that uses hard constraints requires a more careful estimate.

Definition 3 (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 cells 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 $-R \leq \beta \leq 0$ 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 $\mathcal{C}_{\beta=k}$ or simply $\mathcal{C}_k$ (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., $\mathcal{C}_{<-R}$ will denote deep cells. See fig. 5 for illustration.

Algorithm 2: A time step of the band method

1 Transfer velocity from particles to grid **and combine it with the current grid velocity** // $\bar{x}$
2 Apply external forces to grid
3 Project velocity onto divergence-free fields
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' velocities are copied only for cells within the band, not including the band interface. The velocity in the rest of the grid cells remains unchanged. When correcting the particle positions, 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 [31]. Cell markings $\mathcal{C}$ 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 that 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 with every deletion and insertion of a particle. The excess of particles in the deep is

$$n_{\text{excess}} \coloneqq n_{\text{deep}} - \mu|\mathcal{C}_{<-R}| \ .$$

When $n_{\text{excess}} > 0$, we add n_{excess} particles to the band interface. We randomly iterate over 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 because 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 previous one. Foundation and concepts are laid out throughout 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 ($\beta = 1 - R$) into the band interface ($\beta = -R$) to equal the number of particles that move in the opposite direction.

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

$$\begin{aligned}\tilde{\gamma}_{\text{in}} &\coloneqq \left\{(i,j) \mid \text{cell}\left(\bar{x}_j\right) \in \bar{\mathcal{C}}_{1-R} \ , \ \text{cell}\left(\xi_{ij}\right) \in \bar{\mathcal{C}}_{-R}\right\} \\ \tilde{\gamma}_{\text{out}} &\coloneqq \left\{(i,j) \mid \text{cell}\left(\bar{x}_j\right) \in \bar{\mathcal{C}}_{-R} \ , \ \text{cell}\left(\xi_{ij}\right) \in \bar{\mathcal{C}}_{1-R}\right\} \ .\end{aligned}$$

The deep may contain air bubbles from cells that carried bubbles while they moved into it. 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 numbers of cells and particles:

$$\begin{aligned}\alpha_{-R} &\coloneqq \mu|\bar{\mathcal{C}}_{-R}| - \sum_{c \in \bar{\mathcal{C}}_{-R}} |\bar{\gamma}_c| \\ \alpha_{<-R} &\coloneqq \mu|\bar{\mathcal{C}}_{<-R}| - n_{\text{deep}} \ .\end{aligned}$$

We express the conditions above as an additional constraint to eq. (7):

$$0 \leq \sum_{(i,j)\in\tilde{\gamma}_{\text{in}}} b_{ij} - \sum_{(i,j)\in\tilde{\gamma}_{\text{out}}} b_{ij} \leq \alpha_{\leq -R} \ . \tag{8}$$

Here, $\alpha_{\leq -R} \coloneqq \alpha_{-R} + \alpha_{<-R}$. We also update eq. (7d) and eq. (7e) to use the band markings:

$$\sum_{(i,j)\in\bar{\gamma}_c} b_{ij} \leq \mu \ , \quad \forall c \in \bar{\mathcal{C}}_{\text{empty}} \cup \bar{\mathcal{C}}_{\text{surface}} \cup \bar{\mathcal{C}}_{-R} \tag{9a}$$

$$|\bar{\gamma}_c| \leq \sum_{(i,j)\in\bar{\gamma}_c} b_{ij} \leq \mu \ , \quad \forall c \in \bar{\mathcal{C}}_{-R<\beta<0} \ , \tag{9b}$$

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

The correction step is performed as before by solving the updated problem in eq. (7) for b and updating x accordingly. Unlike the local, sparse constraints in eq. (7), the band constraint is global and dense because it encompasses and ties together particle movements along the band interface. Moreover, the system matrix may no longer be totally unimodular, 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. (8) is to first determine 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. (7) as is (without an additional band constraint) to obtain an optimal solution b^*; we do not update x yet. From these particle movements, we denote the number of particles that go into and out of the band interface by

$$\begin{aligned}n^*_{\text{in}} &\coloneqq \sum_{(i,j)\in\tilde{\gamma}_{\text{in}}} b^*_{ij} \\ n^*_{\text{out}} &\coloneqq \sum_{(i,j)\in\tilde{\gamma}_{\text{out}}} b^*_{ij} \ .\end{aligned}$$

Consider the differences

$$\begin{aligned}s_{\text{in}} &\coloneqq n^*_{\text{out}} - n^*_{\text{in}} + \alpha_{\leq -R} \\ s_{\text{out}} &\coloneqq n^*_{\text{in}} - n^*_{\text{out}} \ .\end{aligned}$$

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. Otherwise, there is negative space (incompressibility is violated), and we solve eq. (7) a second time with an additional constraint, depending on which space is negative.

If $s_{\text{in}} < 0$, then too many particles have moved into the band interface, and we must 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 remaining movements in $\tilde{\gamma}_{\text{out}}$. In addition, we prevent movement into the band from particles in $\bar{\mathcal{C}}_{1-R}$ 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 ones that moved into the band in b^*:

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

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

$$\sum_{(i,j)\in\tilde{\gamma}_{\text{in}}} b_{ij} = n^*_{\text{out}} + \alpha_{\leq -R} \ . \tag{10c}$$

When setting a movement b_{ij} of the j-th particle in eq. (10a) to one, by eq. (7c), we can also set the rest of the particle's movements to zero: $\forall k \neq i, b_{kj} = 0$. Note that eq. (10a) and eq. (10b) merely eliminate variables from the system, leaving a single constraint eq. (10c) 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. (7) with the additional constraint eq. (10) is feasible.

Otherwise, $s_{\text{out}} < 0$, and we must limit the number of particles that move out of the band. Similar to eq. (10), this is expressed as

$$b_{ij} = b^*_{ij} \ , \qquad \forall\,(i,j) \in \tilde{\gamma}_{\text{in}} \tag{11a}$$

$$b_{ij} = 0 \ , \qquad \forall\,(i,j) \in \tilde{\gamma}_{\text{out}} \ , \ b^*_{ij} = 0 \tag{11b}$$

$$\sum_{(i,j)\in\tilde{\gamma}_{\text{out}}} b_{ij} = n^*_{\text{in}} \ , \tag{11c}$$

where eq. (11a) fixes the variables of movements into the band interface, eq. (11b) prevents movements out of the band interface that do not occur in b^*, and eq. (11c) sets the number of particles that leave the band interface.

To summarize, in the first step, we solve eq. (7). If needed, we perform a second step, where we solve eq. (7) again using the one-way band constraints in eq. (10) or eq. (11). After the steps, we update x and proceed with the rest of the algorithm. See fig. 6 for an example.

The one-way band constraint is still dense, and the ILP still cannot be relaxed. However, because we reduce the number of variables and simplify the band problem, the revised problem becomes significantly faster to solve than solving with the full band constraint in eq. (8).

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 less expensive 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. (7) 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 have flowed into or out of the band interface. To correct this imbalance, we move some of them along grid paths in the required directions.

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

We limit the j-th particle's movement to a single cell (in a von Neumann neighborhood) relative to its position in the previous time step ($\bar{x}_j$). 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 c 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. (9a)). We need to find n_{move} such paths.

We formulate this as a minimum-cost flow problem (MCFP) on 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 $\mathcal{C}_{\text{source}}$ and $\mathcal{C}_{\text{sink}}$. The locations 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 j-th particle into a cell c is the cost of its (optimal) position in c minus the cost of its current position with respect to its ideal position:

$$\sigma_{\text{edge}}\left(j,c\right) \coloneqq \sigma_{\text{obj}}\left(\phi_{\text{close}}\left(\hat{x}_j, c\right), \hat{x}_j\right) - \sigma_{\text{obj}}\left(x_j, \hat{x}_j\right) \ . \tag{12}$$

We limit the movement of the j-th particle to its cell in the previous time step and to 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 data related to the cells in three arrays (of size $|\mathcal{C}|$): J, b_{fin}, and σ. $J\left[c\right]$ is the index of a particle that represents an edge to the parent of c on a path, where c can belong to at most one path. $b_{fin}\left[c\right]$ is a (boolean) flag that indicates if the path that started at the source cell c is complete. $\sigma\left[c\right]$ 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* (line 6). A *Node* represents the

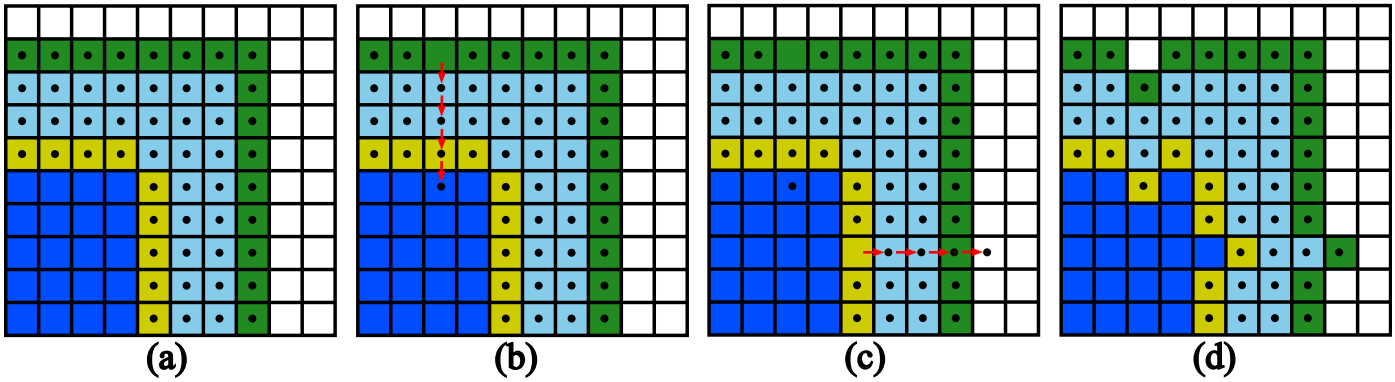


Figure 6: An example of correcting the band (1 ppc). (a) The beginning of the time step; see fig. 5 for the color code of the cells. (b) First step, solving the LP in eq. (7). Four particles move into the cell below them; red arrows indicate their former cells. One of the particles moved into the band 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 must perform a second step to correct this violation. If we use the variant in section 6.2, then we solve eq. (7) again—based on the particle positions in (a)—fixing the movements in eqs. (10a) to (10b) and adding the constraint in *eq.* (10c) 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); intercell 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).

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 already been dequeued and has been assigned a parent. Q can hold *Nodes* of the same cell but with different parents (different possible paths). Only the *Node* with the lowest cost is processed, and the rest are ignored. This is an efficient alternative to 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 available 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 have not been 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 more than one cell away from the particle's cell in the previous time step. It means that either the particle is currently in the same cell as in the previous time step or it is going to move to that cell.

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

Since alg. 3 finds only nonintersecting paths, it may need to 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 has not been 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's algorithm, 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. 6 for an example.

7. Coupling with Solids

We address the incorporation of solids into our framework. We illustrate the idea with a simple scene of an object (also referred to as an obstacle) free falling into water. For clarity and simplicity, we adopt a one-way coupling: The object influences the fluid particles, but the particles do not influence the object's motion. The object experiences only translational dynamics; torque, rotation, shear stresses, and local pressure distribution are neglected. Its trajectory is scripted along a single axis, with either constant speed or acceleration defined by gravity and buoyancy. Buoyancy is prescribed per grid cell based on the object's vertical position in the scene, not on local particle occupancy. The object may pause or continue in the same direction according to the correction rules specified below.

Before the object hits the water, its motion is affected only by gravity. After the object hits the water, the drag

Algorithm 3: Find paths

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

Let b_{fin} be an array of $|\mathcal{C}|$ flags, σ be an array of $|\mathcal{C}|$ costs, and Q be a priority queue
for $c \in \mathcal{C}$ **do** $J[c], b_{fin}[c], \sigma[c] \leftarrow$ none, false, ∞
for $c \in \mathcal{C}_{\text{source}}$ **do** // initialize Q
  if $|\gamma_c| = 0$ **then** continue // no particles
  $a \leftarrow$ Node(cost=0, cell=c, edge=ROOT, root=c)
  Q.enqueue(a) // using a.cost as key
  $\sigma[c] \leftarrow 0$
while not Q.empty() **do** // main loop
  $a \leftarrow Q$.dequeue() // lowest cost
  $c \leftarrow a$.cell
  if $J[c] \notin \{\text{none}, \text{ROOT}\}$ **then** continue // visited
  if $b_{fin}[a.\text{root}]$ **then** continue // finished
  $J[c] \leftarrow a$.edge // assign an edge to the parent
  if $c \in \mathcal{C}_{\text{sink}}$ and $|\gamma_c| < \mu$ **then** // a sink with available space
    $b_{fin}[a.\text{root}] = \text{true}$
    P.add(Path(edge=a.edge, sink=c))
    continue
  for $d \in \mathcal{D} \setminus \{0\}$ **do** // excluding the zero vector
    $c' \leftarrow \text{cell}(\phi_{\text{center}}(c) + d)$
    if $J[c'] \neq$ none **then** continue // visited or a source
    $j \leftarrow$ best_edge(c, c')
    if $j =$ none **then** continue // no edge
    $t \leftarrow \sigma[c] + \sigma_{\text{edge}}(j, c')$ // total cost from the source
    if $t \geq \sigma[c']$ **then** continue // is not better
    $\sigma[c'] \leftarrow t$
    Q.enqueue(cost=t, cell=c', edge=j, root=$a.root$))

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

$\sigma, j \leftarrow \infty$, none
for $j' \in \gamma_c$ **do**
  if $c \neq \text{cell}(\bar{x}_{j'})$ and $c' \neq \text{cell}(\bar{x}_{j'})$ **then** continue // farther than one cell from $\bar{x}_{j'}$
  if $j =$ none or $\sigma_{\text{edge}}(j', c') < \sigma$ **then**
    $\sigma, j \leftarrow \sigma_{\text{edge}}(j', c'), j'$

Algorithm 5: Update a path

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

$j, c \leftarrow r$.edge, r.sink
while $j \neq$ ROOT **do**
  $c' \leftarrow \text{cell}(x_j)$ // parent cell
  $x_j \leftarrow \phi_{\text{close}}(\hat{x}_j, c)$ // move the particle
  $j, c \leftarrow J[c'], c'$ // predecessor

and buoyancy forces come into play, decelerating the object until it reaches terminal velocity. The effect on the fluid is expressed in the boundary conditions of the pressure equation [4, chapter 5]. The pressure in grid cells that are marked as solid is set to $p = 0$, and along the solid boundary we have

$$u \cdot n = u_{\text{solid}} \cdot n \ , \tag{13}$$

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

The position of a moving object is tracked, and its representation (e.g., a mesh or *solid* particles) determines the grid cells it occupies. We allow a (non-empty) grid cell to be occupied by either fluid or a solid, which determines its marking. As with fluid particles, the time-step size is limited to prevent a solid from moving past the neighboring cells (Moore neighborhood). If an object is going to occupy new cells, then the correction step decides whether the object moves or stays in place. For simplicity, we make further assumptions: (i) the objects are denser than the fluid, and (ii) the movement of an object is prioritized unless incompressibility or fluid speed is violated. It means that an object falling into the water should continue moving smoothly to the bottom of the tank, with acceleration determined by gravity and buoyancy alone, while fluid particles clear the way. This movement should be impeded only if it is going to cause unnatural fluid behavior such as overacceleration or compression.

Let $\mathcal{C}_{\text{new\_solid}}$ be the set of cells that are not marked as solid and that the object intends to move into. We modify the objective in eq. (7a) to use a new objective function:

$$\min_{b} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} b_{ij} \sigma_{\text{solid\_obj}}(\xi_{ij}, \hat{x}_j) \ , \tag{14}$$

where

$$\sigma_{\text{solid\_obj}}(q, r) := \begin{cases} \lambda_{\text{penalty}} & \text{if cell}(q) \in \mathcal{C}_{\text{new\_solid}} \\ \sigma_{\text{obj}}(q, r) & \text{else} \end{cases} . \tag{15}$$

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 $\mathcal{C}_{\text{new\_solid}}$; otherwise the object stays in place.

For the band method in section 6.3, we modify σ_{edge} in eq. (12) to use $\sigma_{\text{solid\_obj}}$ instead of σ_{obj}.

The definition of the fluid surface (definition 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

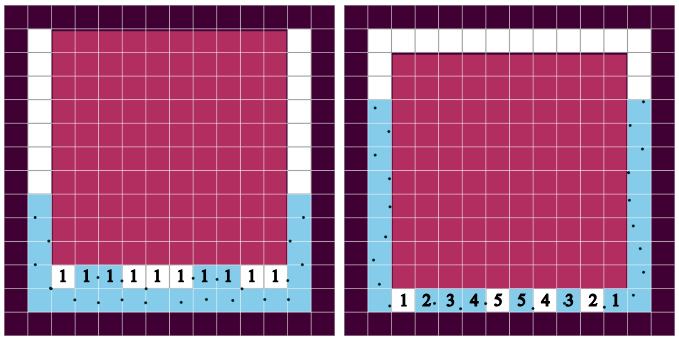


Figure 7: Clearing distance. Two frames of an obstacle (red square) falling into water (1 ppc). The clearing distance of cells in $\mathcal{C}_{\text{new\_solid}}$ is marked. (Left) all the cells in $\mathcal{C}_{\text{new\_solid}}$ have a non-obstacle neighbor below them, and their clearing distance is 1. (Right) Only the particle in the rightmost cell in $\mathcal{C}_{\text{new\_solid}}$ is guaranteed to clear the way using eq. (15). Other particles in $\mathcal{C}_{\text{new\_solid}}$ require eq. (16) to guarantee progress towards clearing the way.

Consider a particle that occupies a potentially new obstacle cell and currently blocks the obstacle movement. As long as there is a path for this particle in the surrounding fluid to a cell with free space not in $\mathcal{C}_{\text{new\_solid}}$, an optimal solution will push it along that path to move it out of the obstacle's way and avoid the penalty in eq. (15).

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 $\mathcal{C}_{\text{new\_solid}}$ 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. (14) would be the same in each case, and nothing would motivate it to clear the way. To address that, similar to definition 3, we define

Definition 4 (clearing distance). *Each cell in $\mathcal{C}_{new\_solid}$ is assigned a clearing distance that represents its discrete distance from a cell that is not in $\mathcal{C}_{new\_solid}$, where solid cells are ignored. The clearing distance is assigned recursively in a breadth-first-search manner. Cells in $\mathcal{C}_{new\_solid}$ with neighboring cells (in a von Neumann neighborhood) that are not in $\mathcal{C}_{new\_solid}$ are assigned 1. Their unassigned neighbors are assigned one level higher, 2, and so on until all the cells in $\mathcal{C}_{new\_solid}$ are assigned a clearing distance.*

We modify eq. (15):

$$\sigma_{\text{solid\_obj}}(q,r) \coloneqq \begin{cases} \lambda_{\text{penalty}} \cdot \text{cdist}(q) & \text{if } \text{cell}(q) \in \mathcal{C}_{\text{new\_solid}} \\ \sigma_{\text{obj}}(q,r) & \text{else} \end{cases}, \tag{16}$$

where $\text{cdist}(\cdot)$ denotes the clearing distance at $\text{cell}(q)$. This penalizes particles according to their clearing distance; see fig. 7 for illustration.

8. Evaluation

We implemented our method as a plugin for MantaFlow [33], using conjugate gradients to solve a Poisson equation. We used Gurobi [10], selecting the dual simplex algorithm without presolve, to solve LP and ILP problems.

Measuring running time. The experiments were conducted on a laptop equipped with an Intel Core i7-9750H CPU (2.6 GHz, 6 cores) and 32 GB of RAM. The running time of FLIP and IDP is dominated by solving a Poisson equation. FLIP solves one for pressure, and IDP solves an additional Poisson equation 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 3) as

$$V_c \coloneqq \begin{cases} 0 & c \in \mathcal{C}_{\beta>0} \\ \min\left(1, \frac{\|\gamma_c\|}{\mu}\right) & c \in \mathcal{C}_{-1\le\beta\le 0} \\ 1 & \text{else} \end{cases}. \tag{17}$$

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

The measure used in [15] is $\min\left(1, \frac{\|\gamma_c\|}{\mu}\right)$ for any cell c. This measure is more favorable to our method because 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 $V \coloneqq \sum_{c\in\mathcal{C}} V_c$ is the total fluid volume in a time step, and V^* is the volume the fluid should occupy. If there is no emitter in the scene, then V^* is simply the initial fluid volume. We report the range of volume percentages $(100\frac{V}{V^*})$ over all the simulation time steps.

We evaluated our method in several scenes described in section 8.1; see the supplementary video for their animations. Some of the figures show selected frames from

scenes in the video. Statistics on volume preservation and running time are summarized in table 1 and table 2. The grid sizes used in the figures and video are those in table 1. We compared our method with IDP [15], FLIP, the narrow-band FLIP [8], and Power PIC [25].

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., $\times 4$ finer than the simulation grid in 2D). We set $\epsilon = 0.1$, $\eta = 1$, $\tau = \frac{1}{2\sqrt[d]{\mu}}$ (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 doing so increased the number of iterations due to lower accuracy. Besides increasing running time, large scaling of the transportation grid resulted in cracks and holes that repeatedly formed and mended in the fluid. Larger values for ϵ and δ disrupted the uniform particle distribution. On the other hand, the effects of using smaller values ranged from the fluid becoming sluggish and exhibiting extremely high energy dissipation to standing still. To summarize, Power PIC can correct the fluid's volume and particle distribution, but it does so at the risk of introducing dissipation when the changes are aggressive.

8.1. Scenes

The default settings 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 only for our method in 3D (the variant in section 6.3). The band thickness is $R = 3$.
- The maximum fluid speed is bounded.

The scenes with an obstacle follow the one-way coupling described in section 7.

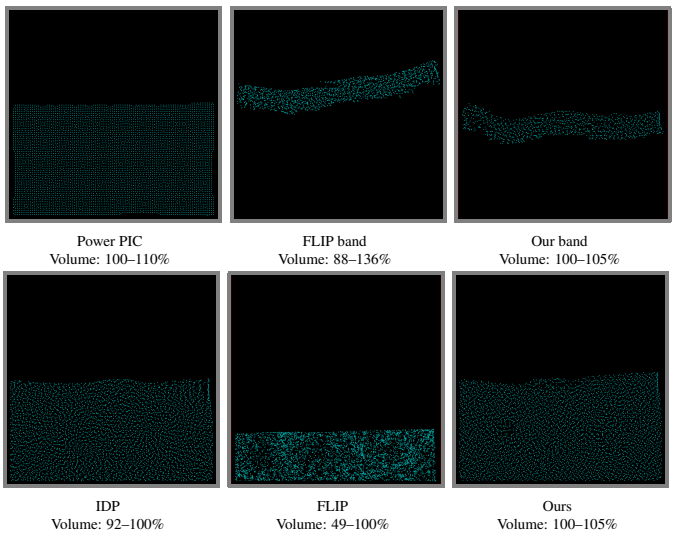


Figure 8: The last frame from a dam scene in the supplementary video.

A breaking dam. In this scene, we perform an initial comparison of the methods' behavior and volume preservation. Some time after 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 half of the tank (domain); see fig. 8.

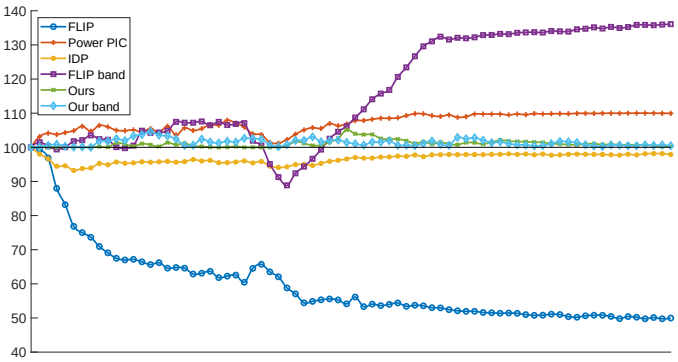


Figure 9: Volume (in percent) over time of the 2D dam scene.

The volume is plotted in fig. 9. FLIP loses volume, and this 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, though 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 with other scenes.

Figure 10 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 11 shows another 2D dam scene (without an emitter) using 1 ppc in a $\times 4$-finer grid (i.e., scaled by two in each dimension). For FLIP and IDP, the fluid collapses, causing 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. 12). 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 at most one particle in a grid cell, there can be no air bubbles, and the volume is perfectly preserved.

Figure 13 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

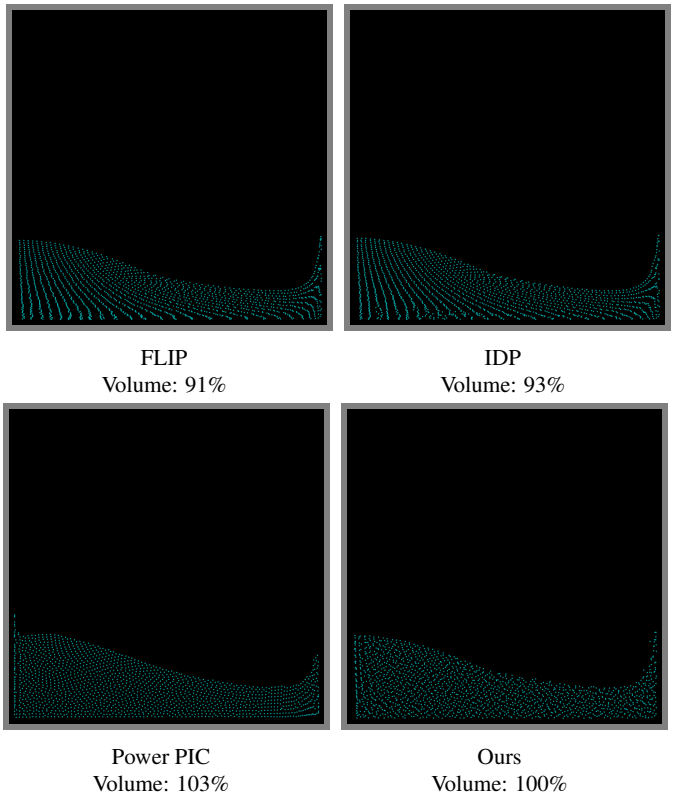


Figure 10: A frame from a dam scene (in the supplementary video), when the water hits the right wall. Volume labels indicate the volume in this frame only.

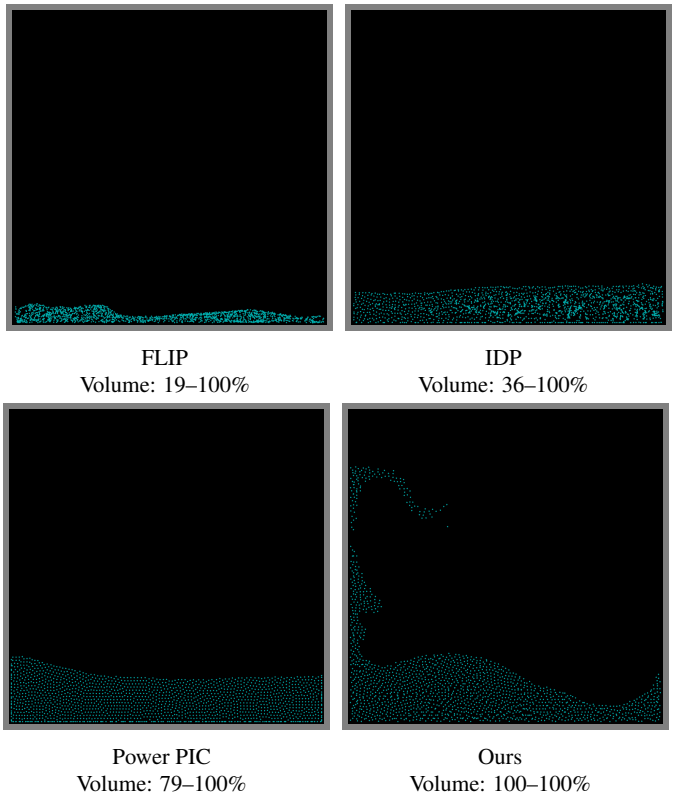


Figure 11: A dam scene with 1 ppc (from the supplementary video).

ours introduces noise similar to the 2D case (fig. 10). Using our method, the fluid hits the left wall at the same time as FLIP does. IDP overshoots the splash, which hits the wall earlier and more strongly. The 1 ppc version of our method requires surface extraction at lower resolution, which is less detailed. Our 8 ppc version gains some volume due to air bubbles.

A drop of water. A drop of water is falling into a pool; see fig. 14. Note where the splash goes. IDP throws the splash off center while ours keeps it centered like FLIP.

Compressing the fluid. A heavy obstacle moves at a constant velocity (scripted) towards the bottom of the tank. Its movement should supersede the fluid's unless fluid speed or incompressibility is 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. 15. An example of a similar real-world scenario is a heavy platen in a sealed hydraulic press or compression rig, where a piston spans the bore and forces the incompressible fluid ahead of it so the platen descends while the fluid is compressed.

We show two options for FLIP and IDP:

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. Moreover, particles still have energy and continue to swirl around even after FLIP finishes. FLIP, on the other hand, lets the obstacle compress the fluid too much, which leads to

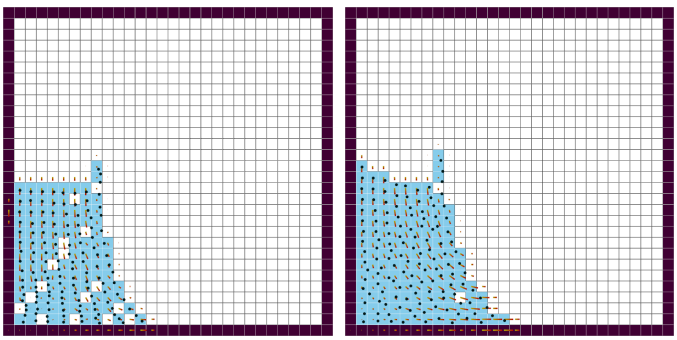
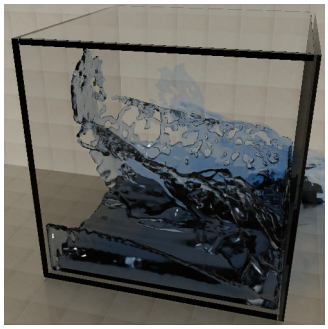


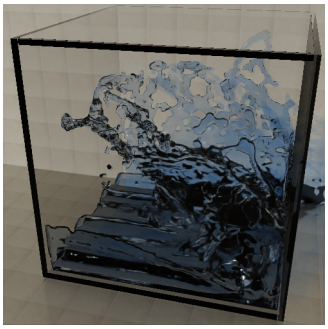
Figure 12: A dam scene on a 30×30 grid with 1 ppc. (Left) FLIP's fluid is riddled with holes that disrupt the velocity field. (Right) Our fluid has fewer holes, and the constraint maintains the volume.

FLIP
Volume: 65–100%



IDP
Volume: 95–100%

Our 1ppc
Volume: 100–100%

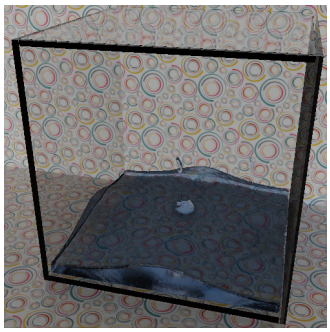


Ours
Volume: 100–108%

Figure 13: A dam scene.

FLIP
Volume: 94–100%

IDP
Volume: 97–100%



Ours
Volume: 100–100%

Figure 14: A water drop.

significant volume loss. For both methods, the obstacle exhibits an undesired halting behavior.

Our method achieves the desired behavior and finishes the simulation much earlier than the other two.

A spiral. In this scene, the plate again compresses the fluid, but here the fluid is forced through a narrow spiral tunnel that channels it into a smaller cavity; see fig. 16. Since the fluid's speed is limited, so is the obstacle's. An example of a similar real-world scenario is a spiral heat exchanger, where fluid is directed through a tightly wound helical channel that forces the liquid into a confined cavity.

We used naive collision detection for FLIP and IDP. FLIP lets the obstacle compress the fluid too much. IDP allows the obstacle to descend 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 permits, and the fluid is compressed as much as the volume restriction permits. After the fluid is compressed as much as possible, particles are clumped inside cells, and the fluid continues to exhibit jerky motion.

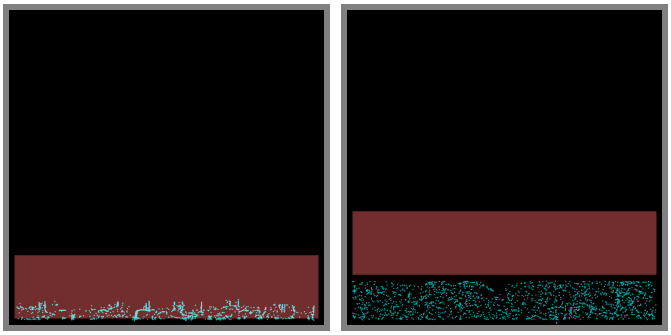
A falling obstacle. An obstacle is falling into the water. An example of a similar real-world scenario is a heavy cargo container dropped into harbor water. When the rigid body enters the free surface, buoyancy and drag slow its descent, and splash dynamics dominate as the surrounding fluid is displaced. 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 does not overlap particles, which would cause volume loss. Figure 17 shows the 2D case, and fig. 18 shows the 3D case.

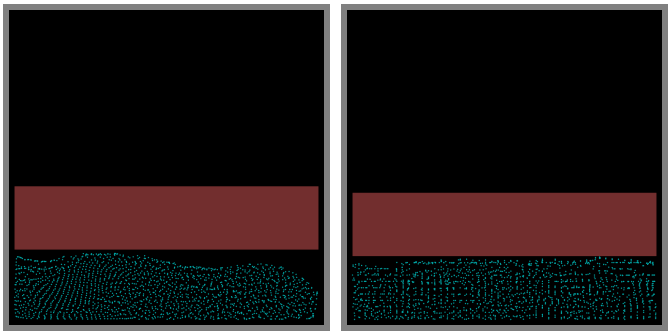
Our method can move the obstacle through the fluid even without the obstacle exerting any force on the fluid. Figure 19 shows an experiment where the boundary conditions for the pressure equation along the obstacle's boundary, eq. (13), are set to zero velocity. While the scene illustrates a specific aspect of our method, an example of a similar real-world scenario is a robotic wedge driven into a block of foam, in which the wedge's motion is prescribed kinematically and the foam itself must deform and rearrange around the obstacle. Because the foam boundaries remain fixed, the obstacle's descent displaces material without active force exchange, illustrating the case of a moving obstacle with zero boundary velocity.

There is nothing to repel the fluid from the obstacle's path. As expected, when FLIP uses collision detection, the



FLIP, no collision detection
Volume: 0–100%

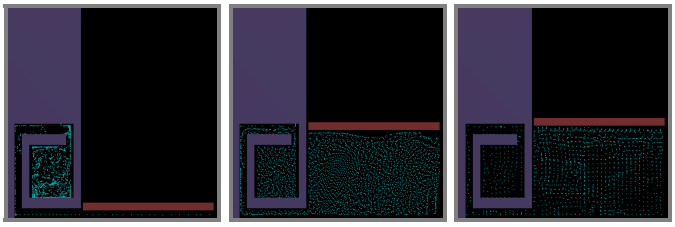
FLIP
Volume: 59–100%



IDP
Volume: 92–100%

Ours
Volume: 100–101%

Figure 15: Compressing the fluid, the final frame.



FLIP
Volume: 17–100%

IDP
Volume: 90–100%

Ours
Volume: 100–100%

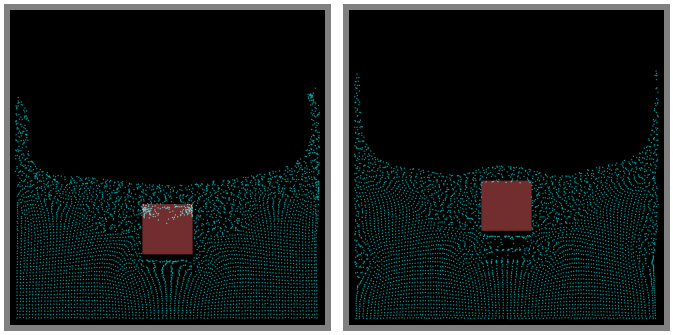
Figure 16: A spiral, the final frame.

obstacle cannot penetrate the fluid. Using our method, the obstacle progresses smoothly through the fluid, and the correction method displaces particles out of the obstacle's path, requiring no other forces.

Figure 20 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 walls. With collision detection enabled 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.

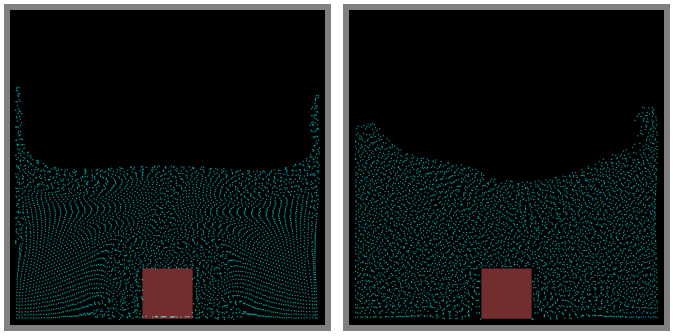
8.2. Discussion

Methods. We focused the experiments on comparing with IDP, whose paper shows comparisons with several other methods. We did not use the band method for FLIP



FLIP
Volume: 90–100%

IDP
Volume: 96–100%



FLIP, no collision detection
Volume: 95–100%

Ours
Volume: 100–100%

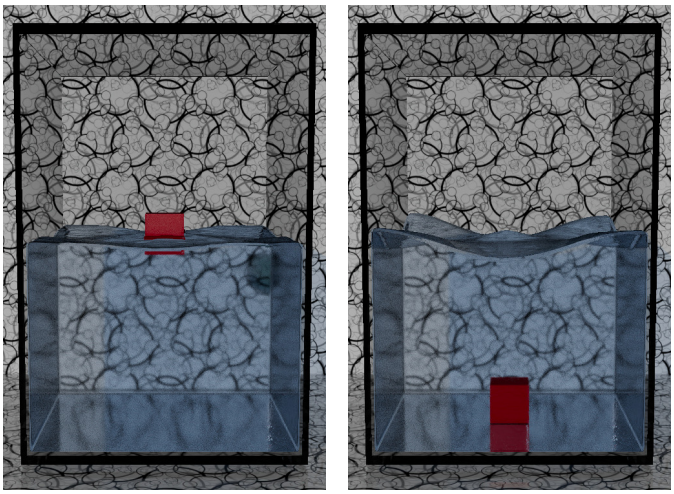
Figure 17: A falling obstacle, the final frame.

to keep the settings close to IDP, which does not support the band method. Also, both methods did not have performance issues that would require it.

Behavior and volume preservation. IDP allows the fluid to violate incompressibility and to 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 from which it cannot be recovered. We proposed several scenarios to challenge this aspect, offering two reasonable solutions for collision detection in FLIP and IDP: ignoring the particles and a naive detection approach. Even if the user manually selects the better of the two for each scenario, none of the behaviors is

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

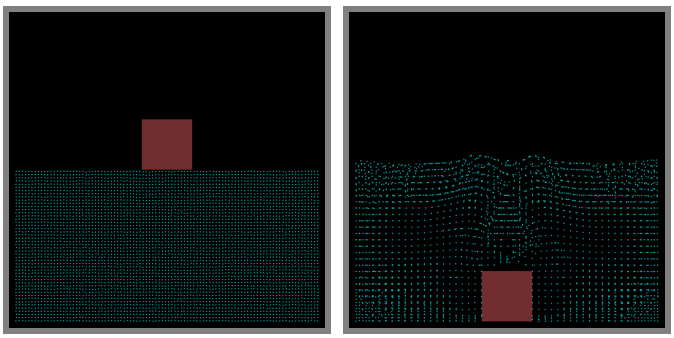
Table 1: Volume preservation. A method's column shows the range of the fluid volume (presented as a percentage of the volume it should occupy) over all the simulation time steps, based on eq. (17).



IDP
Volume: 99–100%

Ours
Volume: 100–100%

Figure 18: A falling obstacle, the final frame. IDP with naive collision detection.



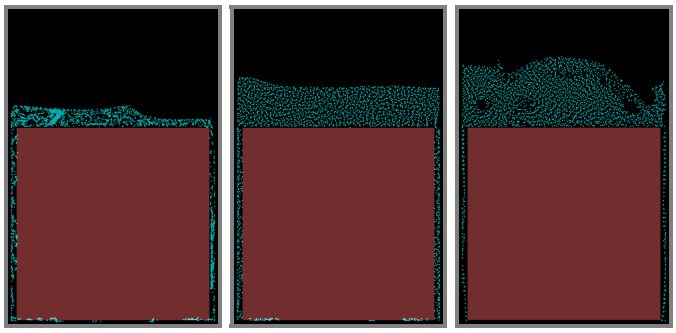
FLIP
Volume: 100–100%

Ours
Volume: 100–100%

Figure 19: A falling obstacle with boundary conditions set to zero velocity, the final frame.

quite acceptable. The naive method caused a halting behavior or even a premature complete stop. Ignoring particles led to 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 rather than 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 thereafter. 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 achieved mostly by blocking particles from moving into neighboring cells rather than by pushing 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 reach. This is also prioritized over a more uniform particle distribution, as in Power PIC, which looks nice in 2D but



FLIP
Volume: 35–100%

IDP
Volume: 71–100%

Ours
Volume: 100–103%

Figure 20: A large falling obstacle, the final frame.

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": the 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. (7), and the MCFP problem in section 6.3. Average quantities are calculated over all time steps. Timings are given in seconds, rounded to one decimal place.

affects the fluid behavior. IDP tends to preserve FLIP's behavior, e.g., fig. 10, where it maintains a smooth surface and clumped particles, whereas the other methods that add noise. However, the corrective movements can also substantially influence fluid behavior, e.g., in the 3D dam when the splash overshoots and hits the left wall (fig. 13) or in the water drop scene, where the splash of water is thrown off the center (fig. 14).

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

The time required to solve 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, consider the extreme scenario in fig. 21. 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 ob-

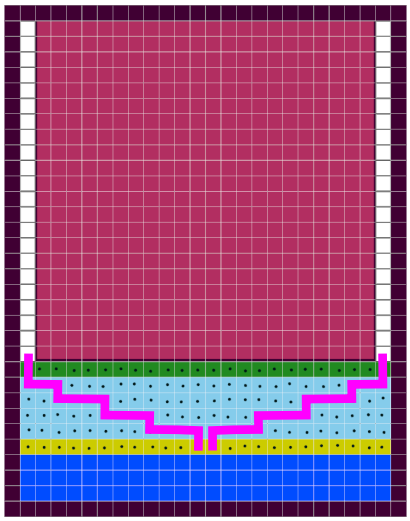


Figure 21: An extreme (hypothetical) case of paths (in magenta) that start in the middle of the band, creating a V shape and blocking other paths.

stacle 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 correlate with the number of particles or grid size. For example, in the first time steps 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 LP objective includes the obstacle, and the solution decides whether the obstacle moves. This creates a dependency between cells in $\mathcal{C}_{\text{new\_solid}}$ 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: 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 by the volume constraint, and it needs to go sideways and upward, which is a more global behavior.

Among the three methods, FLIP is the fastest since it requires only one Poisson solution, whereas the other two methods require additional solves. 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-sized grids. However, in some cases, while solving the MCFP remains reasonably fast, solving the LP can take several minutes, which leaves room for improvement.

Grid artifacts. Our method exhibits grid artifacts. Grid lines can be discerned in 2D, where particles are stopped from reaching neighboring cells. This effect, however, is cosmetic. As shown, it does not affect fluid behavior or surface extraction in 3D.

There is also particle clumping within a cell, especially if more than one particle is halted near a cell boundary. This, however, is not an issue for our method. Eventually, particles may be allowed to progress one at a time and will disperse naturally. As can be seen from the 1 ppc experiments, the fluid behavior and surface extraction differ from the default ppc, i.e., the clumping does not reduce the fluid behavior to 1 ppc.

Consider the extreme case where two particles are clumped to the same position in space and have the same velocity. Other methods have no way to separate them. Our method, however, treats each particle individually via the cell constraints, which will eventually separate them.

Since these grid artifacts are cosmetic, we chose not to address them, which may bias the fluid behavior.

Summary. In standard scenes, such as the breaking dam and the water drop, IDP performs well enough and is easy to implement. In extreme cases, such as compressing the fluid or using 1 ppc, our method has a clear advantage, and it maintains the incompressibility condition strictly. This comes at the cost of additional complexity and running time.

9. Conclusion

We propose 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 demonstrate experimentally that such expansion is moderate. Maintaining 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 and may cause noticeable artifacts. A more severe issue is that the fluid can reach a state that is irrecoverable.

Our framework can be further exploited in other applications; 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 time step, an LP is solved. Besides the number of particles, the fluid configuration affects the running time, which may be longer than desired. We offer acceleration via an adapted version of the band method that enforces incompressibility, and we showed experimentally that it performs reasonably on moderate-sized grids. The fastest variation of our band method solves an easier LP, followed by an additional correction that solves an MCFP. Although the solution is not optimal, the result is reasonable for the affected number of particles. A future avenue could be to find faster alternatives to the LP.

A related limitation is that the simulation time step must be small enough that no particle moves more than one grid cell per step; this is analogous to the CFL condition, CFL ≤ 1, which is commonly enforced in explicit schemes for stability. In graphics applications, however, it is sometimes acceptable to relax this restriction and trade some accuracy for increased speed.

In summary, when volume errors are negligible, IDP is sufficiently effective and straightforward to implement. In extreme cases or if perfection is desired, our method offers strict incompressibility. This, however, comes at the cost of added complexity and running time.

References

[1] Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. "Preserving fluid sheets with adaptively sampled anisotropic particles". In: *IEEE Transactions on Visualization and Computer Graphics* 18.8 (2012), pp. 1202–1214.

[2] Stefan Band et al. "Pressure boundaries for implicit incompressible SPH". In: *ACM Transactions on Graphics* 37.2 (2018), pp. 1–11.

[3] Jan Bender and Dan Koschier. "Divergence-Free SPH for Incompressible and Viscous Fluids". In: *IEEE Transactions on Visualization and Computer Graphics* 23.3 (2017), pp. 1193–1206.

[4] Robert Bridson. *Fluid simulation for computer graphics*. CRC press, 2015.

[5] Jumyung Chang et al. "Curl-Flow: Boundary-Respecting Pointwise Incompressible Velocity Interpolation for Grid-Based Fluids". In: *ACM Transactions on Graphics* 41.6 (2022), pp. 1–21.

[6] Shiyi Chen and Gary D Doolen. "Lattice Boltzmann method for fluid flows". In: *Annual review of fluid mechanics* 30.1 (1998), pp. 329–364.

[7] Fernando De Goes et al. "Power particles: an incompressible fluid solver based on power diagrams." In: *ACM Trans. Graph.* 34.4 (2015), pp. 50–1.

[8] Florian Ferstl et al. "Narrow band FLIP for liquid simulations". In: *Computer Graphics Forum*. Vol. 35. 2. 2016, pp. 225–232.

[9] Dan Gerszewski and Adam W Bargteil. "Physics-based animation of large-scale splashing liquids." In: *ACM Trans. Graph.* 32.6 (2013), pp. 185–1.

[10] Gurobi. *Gurobi Optimizer Reference Manual*. 2018. URL: http://www.gurobi.com.

[11] Markus Ihmsen et al. "Implicit Incompressible SPH". In: *IEEE Transactions on Visualization and Computer Graphics* 20.3 (2014), pp. 426–435.

[12] Chenfanfu Jiang et al. "The material point method for simulating continuum materials". In: *Acm siggraph 2016 courses*. 2016, pp. 1–52.

[13] Byungmoon Kim et al. "Simulation of bubbles in foam with the volume control method". In: *ACM Transactions on Graphics* 26.3 (2007), 98–es.

[14] Dan Koschier et al. "A Survey on SPH Methods in Computer Graphics". In: *Computer Graphics Forum* 41.2 (2022).

[15] Tassilo Kugelstadt et al. "Implicit density projection for volume conserving liquids". In: *IEEE Transactions on Visualization and Computer Graphics* 27.4 (2019), pp. 2385–2395.

[16] Michael Lentine et al. "Simulating free surface flow with very large time steps". In: *Symposium on Computer animation*. 2012, pp. 107–116.

[17] Bruno Lévy. "Partial optimal transport for a constant-volume Lagrangian mesh with free boundaries". In: *Journal of Computational Physics* 451 (2022), p. 110838.

[18] Zhiqi Li et al. "Lagrangian Covector Fluid with Free Surface". In: *ACM SIGGRAPH 2024 Conference Papers*. 2024, pp. 1–10.

[19] Frank Losasso et al. "Two-way coupled SPH and particle level set fluid simulation". In: *IEEE transactions on visualization and computer graphics* 14.4 (2008), pp. 797–804.

[20] Miles Macklin and Matthias Müller. "Position based fluids". In: *ACM Transactions on Graphics* 32.4 (2013), pp. 1–12.

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

[22] Matthias Müller et al. "Position based dynamics". In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 109–118.

[23] Rahul Narain, Abhinav Golas, and Ming C. Lin. "Free-Flowing Granular Materials with Two-Way Solid Coupling". In: *ACM Transactions on Graphics* 29.6 (2010), pp. 1–10.

[24] Linhai Qiu, Yue Yu, and Ronald Fedkiw. "On thin gaps between rigid bodies two-way coupled to incompressible flow". In: *Journal of Computational Physics* 292 (2015), pp. 1–29.

[25] Ziyin Qu et al. "The power particle-in-cell method". In: *ACM Transactions on Graphics* 41.4 (2022).

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

[27] Ritoban Roy-Chowdhury, Tamar Shinar, and Craig Schroeder. "Higher order divergence-free and curl-free interpolation on MAC grids". In: *Journal of Computational Physics* 503 (2024), p. 112831.

[28] Takahiro Sato et al. "Extended narrow band FLIP for liquid simulations". In: *Computer Graphics Forum*. Vol. 37. 2. 2018, pp. 169–177.

[29] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[30] Craig Schroeder, Ritoban Roy Chowdhury, and Tamar Shinar. "Local divergence-free polynomial interpolation on MAC grids". In: *Journal of Computational Physics* 468 (2022), p. 111500.

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

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

[33] Nils Thuerey and Tobias Pfaff. *MantaFlow. http://mantaflow.com*. 2018.

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

[35] Xiao Zhai et al. "Fluid simulation with adaptive staggered power particles on GPUs". In: *IEEE Transactions on Visualization and Computer Graphics* 26.6 (2018), pp. 2234–2246.

[36] Yongning Zhu and Robert Bridson. "Animating sand as a fluid". In: *ACM Transactions on Graphics* 24.3 (2005), pp. 965–972.

Appendix A. Proofs

Proposition 1. *The LP relaxation of the ILP in eq. (7), which uses continuous variables, has the same optimal solution.*

Proof. Transform the LP in eq. (7) into the canonical form $\max\{cb \mid Ab \leq d\}$, where A is a matrix, and c, b, 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).

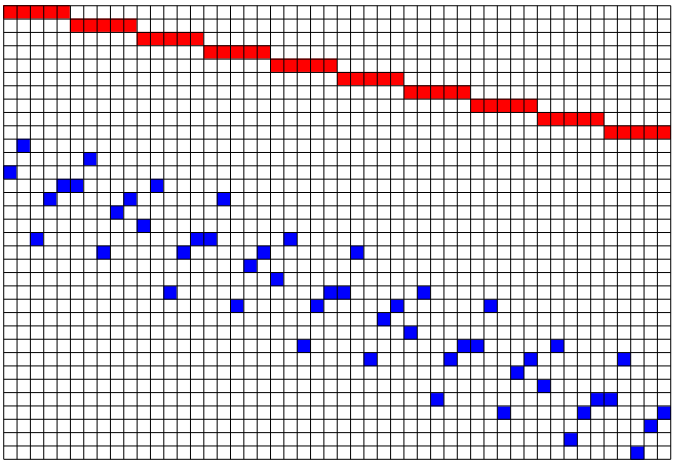


Figure A.22: Visualizing a TU matrix. Zeros are in white, the rest are ones. The rows with the red cells correspond to eq. (7c). The rows with the blue cells correspond to eqs. (7d) to (7f) (without duplicate rows). In both the red and blue set of rows, each column sums up to one.

- 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. (7e) 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) lies 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 [29, theorem 19.1]. A matrix is TU if each of its subdeterminants is $\in \{0, \pm 1\}$.

Lemma 1. *A is TU.*

Proof. The proof is 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 $\{0, \pm 1\}$, and 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 entry B_{ij} (e.g., eq. (7b)). 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 entry, the Laplace expansion along that column reduces the determinant to a cofactor whose value is in $\{0, \pm 1\}$.

- Each variable corresponds to a particle movement, which ends up in a specific cell. Therefore, each variable appears only once in eqs. (7d) to (7f). Moreover, each variable appears once in eq. (7c). 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 submatrices

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

such that a column in each matrix has a single 1; see fig. A.22 for illustration. Let $v \in \mathbb{R}^k$ be the vector

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

where $\mathbb{1}$ denotes a vector of ones. The vector v lies in the null space of $B^\intercal$; hence $\det B = 0$.

□