Preserving Fluid Sheets with Adaptively Sampled Anisotropic Particles

Ryoichi Ando, Nils Thürey, and Reiji Tsuruno

Abstract—This paper presents a particle-based model for preserving fluid sheets of animated liquids with an adaptively sampled Fluid-Implicit-Particle (FLIP) method. In our method, we preserve fluid sheets by filling the breaking sheets with particle splitting in the thin regions, and by collapsing them in the deep water. To identify the critically thin parts, we compute the anisotropy of the particle neighborhoods, and use this information as a resampling criterion to reconstruct thin liquid surfaces. Unlike previous approaches, our method does not suffer from diffusive surfaces or complex re-meshing operations, and robustly handles topology changes with the use of a meshless representation. We extend the underlying FLIP model with an anisotropic position correction to improve the particle spacing, and adaptive sampling to efficiently perform simulations of larger volumes. Due to the Lagrangian nature of our method, it can be easily implemented and efficiently parallelized. The results show that our method can produce visually complex liquid animations with thin structures and vivid motions.

Index Terms—physically based modeling, liquid simulation, Fluid-Implicit-Particle method, thin fluid sheets, adaptive sampling.

1 Introduction

THIN features of liquid are familiar phenomena that we encounter in our daily life, for example the water blade of a small waterfall or the water crown induced by a rain drop. These thin features are important factors for believable visual representations of fluids. Despite of the huge amount work in the area of fluid simulation in computer graphics, thin sheets are still challenging to track precisely. Typically, the discretization resolution severely limits the amount of detail that can be represented. As a result, in Eulerian approaches, such as levelset based simulations, these sheets are quickly filtered out due to the numerical diffusion. In Smoothed-Particle-Hydrodynamics (SPH) simulations, on the other hand, thin sheets often break up into individual particles.

In the last few years, there has been a growing interest in simulating liquids with mesh-based surface tracking methods, e.g., by [1], [2], [3] and [4]. In these methods, surfaces are modeled with explicit triangle meshes, and advected through the underlying Eulerian flow. In contrast to implicit approaches, these approaches do not suffer from numerical diffusion, and can capture details smaller than a grid cell size. However, a drawback of mesh-based surface tracking methods are the difficulties of handling topology changes; the surface meshes must be resampled carefully to avoid tangling whenever topology change events are encountered. The re-meshing may be done by replacing tangled meshes with Marching Cube [5] surfaces, or by detecting vertices that are close to each other and stitching them before the collision takes place. Due to the intricacy of mesh connectivity, the sewing algorithms tend to be complex to implement, and can lead to a loss of detail at the surface.

Since the introduction of the SPH method by Müller et al. [6] to the field of computer graphics, particle-based methods have become popular due to their ease of implementation and their suitability for interactive applications [7], [8], [9]. These particle-based methods are feasible for animations of splashes, although the simulation can suffer from oscillations due to compressibility. Because such oscillations can easily break up thin structures, these methods are not well-suited for tracking very thin fluid features. Besides SPH, researchers proposed various Eulerian-Lagrangian hybrid approaches, such as [10], [11], [12]. These methods are designed to exploit both the advantages of particles and grids. However, as they still rely on particles to represent thin sheets, these can quickly rupture due to the nature of the Lagrangian representation.

Our simulations are based on the commonly used FLIP [11] algorithm, which is a hybrid grid-based method that makes use of particles to represent the liquid volume and increase the accuracy of the advection. FLIP is capable of producing incompressible and turbulent fluid motion that yields interesting and believable visual behavior. We found this method to be a very suitable basis for computing smooth flow fields that lead to the development of thin fluid sheets.

The main highlight of our method is a purely meshless particle-based framework that preserves thin fluid sheets. The thin sheets are preserved by inserting new particles at sparse thin regions in the sheets. These particles are removed as they move away from the

- *R. Ando is with Graduate School of Design, Kyushu University, Japan. E-mail: and@verygood.aid.design.kyushu-u.ac.jp.*
- *N. Thürey is with ScanlineVFX, Munich, Germany. E-mail: nils.thuerey@scanlinevfx.com.*
- *R. Tsuruno is with Faculty of Design, Kyushu University, Japan. E-mail: tsuruno@design.kyushu-u.ac.jp.*

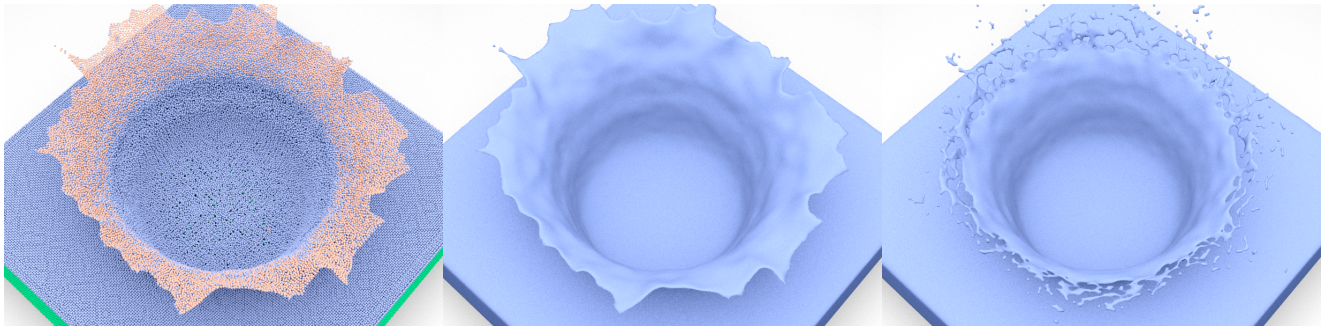


Fig. 1. **Water splash.** Left: A splash with newly inserted particles created by our approach highlighted in red. Middle: The same simulation with a thin surface generated by anisotropic implicit functions. Right: The same simulation frame without our thin sheet preservation.

visible surface. An example of a thin sheet induced by a drop of liquid impacting a surface can be seen in Figure 1. Our work makes substantial use of the anisotropic kernels proposed by Yu and Turk [13]. This algorithm computes a weighted PCA for neighboring particles to estimate their stretch and orientation. This information is used to reconstruct detailed and smooth surfaces. We additionally use it as criterion to judge whether the particles are in the bulk volume or whether they are part of a thin sheet. For particles in a thin sheet we propose a resampling algorithm that preserves the surface of the thin sheet and ensures a regular particle spacing while modifying the dynamics of the simulation as little as possible.

In the original FLIP method, the distribution of particles can become unbalanced over time. Such uneven distributions can be responsible for holes in the fluid, and unwanted changes in volume. We address this issue by incrementally adding an anisotropic displacement to particle positions, to maintain a uniform particle distribution. This process can be interpreted as a weak particle remeshing step. This has advantages when applying our algorithm for the adaptive particle resampling. A two-dimensional example of an adaptive simulation with our approach can be seen in Figure 2. In addition, we describe a simple method for handling accurate boundary conditions with solid walls and free surfaces.

To reduce the computational cost of neighboring particle lookups, we adaptively replace particles in the deep water with fewer large ones. This reduces the overall number of particles and increases the efficiency of our simulations. At the visible surface, we keep small particles to allow for a detailed representation of the surface.

2 Previous Work

Our work draws upon a broad range of research in fluid animation, including particle-based liquid animations and surface tracking methods applied to fluids. In this section, we briefly review previous works closely related to our approach.

State-of-the-art surface tracking for liquid can be categorized into two approaches: implicit and explicit methods. The level set method proposed by Osher and Sethian [14] is one of the most popular implicit methods. In the level set method, each grid node is assigned a distance function from the closest surface position, and the surface is implicitly located where the function is zero. The method has been refined and revised in many ways. Enright et al. [15], Wang et al. [16], and Mihalef et al.[17] placed Lagrangian particles to reduce numerical dissipation. Bargteil et al.[18] updated the signed distance field from a reconstructed surface mesh to increase the accuracy. Heo and Ko [19] preserved the surface detail with a spectrally refined level set (SRL) and a high-order re-initialization method. These implicit methods can be utilized to increase the overall detail with a high resolution grid. However, to the best of our knowledge, such methods cannot completely prevent the rupture of thin surfaces. In addition, some researchers used seeded particles to produce splashes [20], [21], [22] or fluid sheets [23].

Among the explicit approaches, Hirt and Nichols [24] proposed a *volume-of-fluid* method that uses a proportion of the interface for the entire cell. This method is seldomly employed due to the difficulties of handling a discontinuous interface. In explicit methods, Lagrangian approaches are often preferred.

Over the past two decades, a number of mesh-based surface tracking methods have been proposed through a variety of research fields, such as medical image analysis and fluid dynamics [2], [4], [25], [26], [27], [28], [29]. Typically, a mesh-based surface tracker advects explicit surface elements by the underlying motion; however, this method can suffer from self-intersection or complex topology changes. These issues may be resolved by resampling meshes only where collisions take place. This strategy tends to make the algorithm complex because it needs to tolerate numerous complex situations. We want to emphasize that our method is not related to this family of surface tracking methods. On the contrary, our particle-based approach does not hold any connectivity information. Thus, our method does not suffer from such complexities.

Particle methods such as the SPH method [6], [30],

[31], the Moving Particle Semi-implicit (MPS) method [32], [33], and the meshless physical simulations [34], [35], [36] are usually used not only to simulate entire physical motion, but also to reconstruct surfaces. Blinn[37], Zhu and Bridson[11], Adams et al.[38], and Yu and Turk [13] proposed a novel surface reconstruction algorithm from a point cloud. In their method, surfaces are implicitly defined with respect to the distance from particles. However, these methods exhibit poor, blobby ruptures where the particles are sparse. In our particle-based method, we employ the method of Yu and Turk [13] to reconstruct surfaces after filling ruptures with extra particles so that the sheets are preserved. As an underlying fluid solver, we use a modified version of the hybrid grid and particle based approach proposed in [11].

The adaptive resampling phase in our algorithm for deep water volumes is considered to be a variation of adaptive particle-based methods, such as [38], [39], [40], [9], in which the particles are resampled using different sizes to reduce the computational cost while retaining visual detail. On the other hand, in our model for thin sheet preservation, we specifically focus on keeping the continuous fluid sheets instead of increasing the efficiency.

The combination of the SPH and FLIP methods was recently demonstrated by Raveendran et al.[41]. The goal of their approach is to transfer the pressure values enforced on a grid onto the particles to predict the initial guess of SPH pressures. In contrast, we weakly displace particles to keep uniform particle distances. Both methods share the notion of pressure based forces, but we specifically use weak kernels, and base our simulations on an extended FLIP solver.

3 Modified FLIP Solver

This section describes an underlying liquid solver which we used to generate smooth liquid motion for preserving thin surfaces. Our solver is a modified version of the Particle-in-Cell/Fluid-Implicit-Particle (PIC/FLIP) method [11], which is an Eulerian-Lagrangian hybrid approach in which a liquid domain is discretized with a collection of particles. We describe the workflow in detail in the following subsections.

3.1 Interpolation

In contrast to the standard FLIP method, we make use of SPH-like interpolation kernels to interpolate particle based properties, such as velocities. We use this interpolation both for the underlying simulation as well as for our thin sheet preservation algorithm later on. Given a particle distribution, the fluid velocity $\boldsymbol{u}$ and the particle density ρ at location $\boldsymbol{x}$ are computed as

$$\boldsymbol{u}(\boldsymbol{x}) = \frac{\sum_i m_i \boldsymbol{u}_i W_{sharp}(\boldsymbol{p}_i - \boldsymbol{x}, \alpha_u d_i)}{\sum_i m_i W_{sharp}(\boldsymbol{p}_i - \boldsymbol{x}, \alpha_u d_i)} \quad (1)$$

$$\rho(\boldsymbol{x}) = \sum_i m_i W_{smooth}(\boldsymbol{p}_i - \boldsymbol{x}, \alpha_\rho d_i), \quad (2)$$

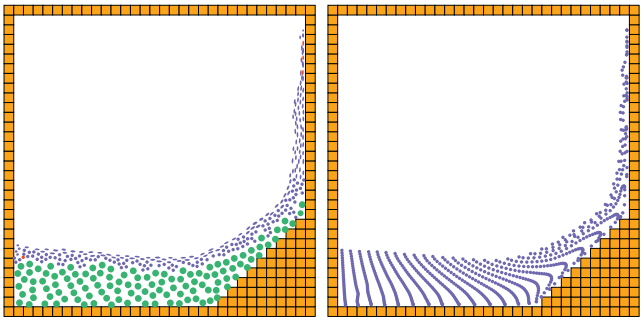


Fig. 2. **Two-dimensional dam break.** Left: Our liquid simulator. Right: A normal PIC/FLIP simulation. Notice that in our image particles are adaptively sampled in the deep water, and the spatial particle distribution is almost uniform while the PIC/FLIP image is uneven.

where d_i, $\boldsymbol{p}_i$, $\boldsymbol{u}_i$ and m_i denote the particle radius, the position, the velocity and the mass of a particle i, respectively. α_u and α_ρ are the scaling constants for radius d_i. We used $\alpha_u = 1.0$ and $\alpha_\rho = 4.0$. For weighting kernels we use two simple kernels:

$$W_{sharp}(\boldsymbol{r}, h) = \begin{cases} h^2/||\boldsymbol{r}||^2 - 1 & 0 \le ||\boldsymbol{r}|| \le h \\ 0 & \text{otherwise,} \end{cases} \quad (3)$$

$$W_{smooth}(\boldsymbol{r}, h) = \begin{cases} 1 - ||\boldsymbol{r}||^2/h^2 & 0 \le ||\boldsymbol{r}|| \le h \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

Our sharp kernel $W_{sharp}(\boldsymbol{r}, h)$ is designed to resample a quantity of the particle such that $\boldsymbol{u}(\boldsymbol{p}_i) = \boldsymbol{u}_i$, while $W_{smooth}(\boldsymbol{r}, h)$ is used to compute the average of a quantity from nearby particles. Any kernels, e.g. those commonly used for SPH simulations, with similar properties would be applicable here. To accelerate the neighborhood lookup, we bin the particles into the grid before computing each step of the simulation. If the denominator of Equation 1 is close to zero, we use the velocity of the closest particle to prevent roundoff errors. For each grid cell, we additionally compute a level set function to locate the position of the free surface:

$$\phi_{\mathrm{L}}(\boldsymbol{x}) = \alpha_L N_0 \rho_0 - \sum_i \rho(\boldsymbol{p}_i), \quad (5)$$

where i and ρ_0 denote the particle index within a cell and the initial maximum density at the beginning of the simulation. α_L is a parameter to distinguish incompressible liquid volumes and splashing particles, while N_0 denotes the initial number of particles placed in a cell. The liquid domain consists of all cells with $\phi_{\mathrm{L}}(\boldsymbol{x}) \le 0$. We consistently use $\alpha_L = 0.2$ and $N_0 = 8$ in our three-dimensional simulations. Note that this implicit function is not used for reconstructing thin surfaces. We describe another implicit function to this end in Section 5.3.

The purpose of the implicit functions from Equation 5 is to avoid cells being erroneously flagged as liquid in areas where particles are sparse, such as liquid sheets or gaps in the liquid domain. This prevents liquid particles from floating on the free surfaces, and avoids unwanted volume increase.

3.2 Grid-based Solving

In the first step of our simulation, the velocity of the advected particles is mapped onto a staggered Marker-And-Cell (MAC) grid. The velocity is made divergence free by solving the standard Poisson equation on the grid. In PIC [42], the projected grid velocity is mapped to the particles to yield an incompressible particle flow. In our implementation, we use Equation 1 to map the velocity from the particles to the grid similarly to [43], [39], [40]. Although grid-based interpolation, e.g., as in [11], could be used here, we prefer the use of an SPH-like interpolation since we can consistently use this approach for interpolating arbitrary physical quantities as well as for our position-based correction algorithm from Section 5.2. To map velocity from the grid to the particles, we perform a tri-linear interpolation.

PIC naturally carries the momentum using the particles, instead of performing a grid-based advection step. However, PIC is known to create numerical diffusion due to the successive back-and-forth interpolation of velocities. For this reason, FLIP was introduced by [44]. In FLIP, only the change of the grid velocity from the previous time step to the current one is mapped back to the particles. This delta is added to the velocities of the particles to obtain the new particle velocity for the following time step.

The particles are then moved using the velocity field on the grid. Unlike PIC, FLIP does not suffer from numerical dissipation. However, FLIP can suffer from noisy behavior due to a complete lack of viscosity in the simulation. As in [11], we alleviate this problem by linearly blending the PIC and FLIP velocities with a scalar parameter α_ν as follows:

$$\boldsymbol{u} = \text{PIC/FLIP}(\boldsymbol{u^*}) = \alpha_\nu \text{ FLIP}(\boldsymbol{u}^*)+(1-\alpha_\nu)\text{PIC}(\boldsymbol{u}^*). \quad (6)$$

where $\boldsymbol{u}^*$ and $\boldsymbol{u}$ denote the velocity of particles after advection and the new particle velocity, respectively. Note that α_ν effectively controls the viscosity of the simulation, and is typically close to 1. In our implementation we have used $\alpha_\nu = 0.95$.

3.3 Boundary Conditions

A regular, voxelized pressure solver on a cartesian MAC grid cell can be responsible for stair-casing artifacts at the free surface and particles getting stuck on curved solid walls. This problem is well recognized among researchers, and several approaches exist to prevent this [45], [46]. In the level set community, the ghost fluid method [47] is commonly used for second-order accurate free surface boundary conditions. Let $p(\boldsymbol{x}_i)$ be a pressure at cell i underneath a liquid surface, and $p_\text{G}(\boldsymbol{x}_j)$ an adjacent ghost pressure at cell j outside the liquid volume. We use Equation 5 to compute the correct ghost pressure value as:

$$p_\text{G}(\boldsymbol{x}_j) = \frac{\phi_\text{L}(\boldsymbol{x}_j)}{\phi_\text{L}(\boldsymbol{x}_i)} p(\boldsymbol{x}_i) \quad (7)$$

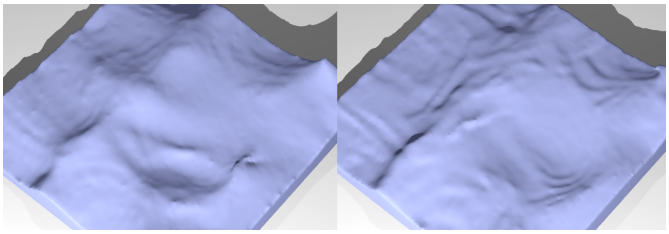


Fig. 3. **Smooth surface.** Left: Surfaces with accurate free surface boundary conditions. Right: Surfaces with voxelized free surface boundary conditions. Notice that in the left image, wrinkle artifacts are less visible.

The effect of the free surface boundary condition is illustrated in Figure 3. As seen from the figure, ghost pressures on the surfaces act to flatten the jagged ripples. For curved solid walls, we used the variational framework introduced by Batty et al. [48] to accurately handle smooth obstacle surfaces. To summarize, our simulations use a FLIP based fluid solver with custom interpolation kernels and second order boundary conditions for the free surface and obstacles. The pseudocode code of our modified solver is shown in Algorithm 1. Here, I denotes the tri-linear interpolation from the grid based velocity g to the particles.

Algorithm 1 MODIFIED FLIP SOLVER

1: $\boldsymbol{u}^* =$ Advect particles with $\boldsymbol{g}_t$
2: $\boldsymbol{g}^* =$ Interpolate $\boldsymbol{u}^*$ using Eq. 1
3: $\boldsymbol{g}_{t+1} = \text{Project}(\boldsymbol{g}^*, \phi_\text{L})$
4: $\boldsymbol{u}_{t+1} = \alpha_\nu[\boldsymbol{u}^* + I(\boldsymbol{g}_{t+1} - \boldsymbol{g}^*)] + (1-\alpha_\nu)I(\boldsymbol{g}_{t+1})$

4 PRESERVING SHEETS

The main highlight of our method is an algorithm to preserve thin sheets by inserting extra particles where the surface might break up. Figure 4 illustrates an overview of our procedure. First, we extract thin fluid regions. Within those regions, new candidate positions for particles are computed and new particles are carefully inserted to avoid collisions. We describe these steps in more detail in the following subsections.

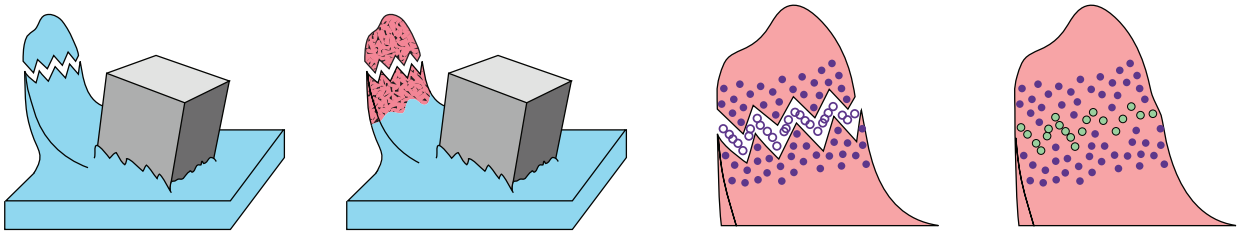
4.1 Thin Particle Extraction

We extract so-called *thin particles* (Figure 4b), which form thin regions that could potentially break up, by examining the stretch of the distributions of neighboring particles. To do this, we employ the anisotropic kernel method of [13]. For each particle, we compute a weighted average covariance of particles C as follows:

$$C_i = \frac{\sum_j (\boldsymbol{p}_j - \overline{\boldsymbol{p}_i})(\boldsymbol{p}_j - \overline{\boldsymbol{p}_i})^T W_{smooth}(\boldsymbol{p}_j - \overline{\boldsymbol{p}_i}, \alpha_\rho d_0)}{\sum_j W_{smooth}(\boldsymbol{p}_j - \overline{\boldsymbol{p}_i}, \alpha_\rho d_0)}, \quad (8)$$

where

$$\overline{\boldsymbol{p}}_i = \frac{\sum_j \boldsymbol{p}_j W_{smooth}(\boldsymbol{p}_j - \boldsymbol{p}_i, \alpha_\rho d_0)}{\sum_j W_{smooth}(\boldsymbol{p}_j - \boldsymbol{p}_i, \alpha_\rho d_0)}. \quad (9)$$



(a) Input Simulation (b) Thin Sheet Area Detection (c) Candidate Position Detection (d) Particle Insertion

Fig. 4. **Algorithmic overview.** (a) Given an input collection of particles, (b) we compute particles in thin regions. To fill in breaking sheets, (c) we compute the candidate positions for insertion and (d) we insert the particles with a suitable spacing.

The Singular Value Decomposition (SVD) of the associated C_i yields the direction and the stretch of neighboring particle positions as eigenvectors and eigenvalues, as follows:

$$C_i = R \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \sigma_3 \end{bmatrix} R^T \tag{10}$$

where R and σ_n denote the eigenvector matrix and the eigenvalues in order of descending magnitude ($\sigma_1 > \sigma_2 > \sigma_3$). The eigenvalues indicate the degree of stretch. Note that the SVD is applied only where the particle density falls below a chosen threshold $\rho(\boldsymbol{p}_i) < \alpha_{\partial\Omega}\rho_0$; otherwise, we skip the SVD, and use $\mathrm{diag}(\sigma_1, \sigma_2, \sigma_3) = \boldsymbol{I}$ instead. Since the SVD is only computed near the liquid surface, this can be done very efficiently. The thin particles are identified by:

$$\sigma_3 \leq \beta_{\mathrm{thin}}\sigma_1, \tag{11}$$

where β_{thin} denotes a threshold that determines the degree of thinness. In our implementation we used $\beta_{\mathrm{thin}} = 0.2$, and $\alpha_{\partial\Omega} = 0.7$. These thin particles are used to compute where new particles are inserted in the following steps.

4.2 Finding the Candidate Positions

Within the extracted thin particles, we search for pairs $(\boldsymbol{p}_i, \boldsymbol{p}_j)$ that bridge breaking holes in the fluid volume. A midpoint of the pair $(\boldsymbol{p}_i+\boldsymbol{p}_j)/2$ is recorded as a candidate position for insertion (Figure 4c). In our method we choose pairs $(\boldsymbol{p}_i, \boldsymbol{p}_j)$ that satisfy all of the following conditions:

$$\begin{cases} \beta_{\mathrm{min}}d_0 \leq ||\boldsymbol{p}_j - \boldsymbol{p}_i|| \leq \beta_{\mathrm{max}}d_0 \\ \sum_k W_{smooth}((\boldsymbol{p}_i + \boldsymbol{p}_j)/2 - \boldsymbol{p}_k, \beta_{\mathrm{min}}d_0) = 0 \\ (\boldsymbol{p}_j - \boldsymbol{p}_i) \cdot (\boldsymbol{u}_j - \boldsymbol{u}_i) > 0 \end{cases} \tag{12}$$

where β_{min} and β_{max} denote constants that control the minimum and maximum space of candidate positions, respectively. In the first row of Equation 12, we check whether the two particles lie at a moderate distance from each other. Here, we use $\beta_{\mathrm{min}} = 0.8$ and $\beta_{\mathrm{max}} = 3.5$. In the second row, we check whether the candidate midpoint is sparse enough to contain a new particle at radius $\beta_{\mathrm{min}}d_0$. Note that at this time, scarcity is only checked among existing particles. Finally, the third row checks whether the pair has opposing velocities. This means that the distance between the pair of particles is currently increasing. Once these pairs are found, we store the midpoints in a list S. We illustrate an example of candidate positions with and without thin particle detection in Figure 5.

4.3 Particle Insertion and Removal

The candidate positions cannot be used naively because they are usually very dense. To prevent the creation of unnecessarily large amounts of particles, we describe a simple algorithm to insert candidates with a suitable sparseness. Let I be the final list of candidates positions. The first entry I_1 is the most sparse candidate within S, given by the lowest density:

$$I_1 = \arg\min_{j \in S} \rho(\boldsymbol{p}_j). \tag{13}$$

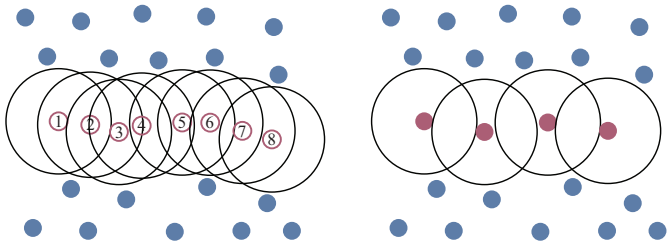
Once I_1 is found, we remove candidates near $\boldsymbol{p}_{I_1}$ from S that exists within radius $\beta_{\mathrm{min}}d_0$. In the second step we search for particles in the neighborhood of $\boldsymbol{p}_{I_1}$ in S. Let N_{I_1} denote the set of particles within radius $\beta_{\mathrm{max}}d_0$ of $\boldsymbol{p}_{I_1}$. Now we insert the closest candidate from N_{I_1} into I. This candidate position is computed with

$$I_2 = \arg\min_{j \in N_{I_1}} ||\boldsymbol{p}_j - \boldsymbol{p}_{I_1}||. \tag{14}$$

We can formulate this search recursively as $I_{n+1} = \mathrm{search}(I_n)$. If the next search fails, meaning that no particle exists in the neighborhood of the previously inserted one, Equation 13 is used instead. Figure 6 shows an example of our insertion procedure. In this example, the candidate positions are computed with the desired spacing.

When inserting new particles at the candidate positions, every attribute is linearly interpolated from the source pair except for the mass. In the case of splitting a pair $(\boldsymbol{p}_i, \boldsymbol{p}_j)$, the mass of the new particle is given by $m = (m_i + m_j)/3$. After the split, the masses of particles i and j are reduced to $m_i^{\mathrm{new}} = \frac{2}{3}m_i, m_j^{\mathrm{new}} = \frac{2}{3}m_j$.

Fig. 5. **2D thin particle clipping.** Left: Simulation input. Right: Both the slashed and the open particles satisfy Eq. 12. Only the slashed particles also satisfy Eq. 11.



(a) Candidate Particles (b) Inserted Particles

Fig. 6. **3D particle insertion (sheet viewed from the top).** (a): Suggested candidate particles. (b): Actually inserted particles and insertion conducted in serial order by the numbers shown. A black contour indicates the particle radius.

This is necessary to evenly distribute mass among the newly created particles for interpolation with Equations 1 and 2. Note that in our model the mass is only used as an interpolation weight for physical quantities. The particles themselves move like massless markers when they are advected through the velocity grid. It should also be noted that adaptively merged particles, which are introduced in Section 5.1, are not split to avoid unnecessary complexity.

In contrast, an inserted particle $\boldsymbol{p}_i$ is removed again once it satisfies at least one of the two following conditions

$$\text{any} \begin{cases} \rho(\boldsymbol{p}_i) > \beta_{\max\rho}\rho_0 \text{ and } \sigma_3 \geq \beta_{\text{thin}}\sigma_1 \\ \text{for any particle } j \;\; ||\boldsymbol{p}_i - \boldsymbol{p}_j|| < \beta_{\text{dist}} d_0 \end{cases} \tag{15}$$

where $\beta_{\max\rho}$ and β_{dist} denote the maximum density coefficient and the minimum space, respectively. For our simulations we used $\beta_{\max\rho} = 0.2, \beta_{\text{dist}} = 0.2$. Note that the algorithm so far can introduce flickering once a newly inserted particle is removed in the subsequent time step. To prevent this, we do not remove particles unless they have existed for a minimal number of three successive time steps. When removing a particle, its mass is simply returned to the source pair that was used to generate it. Note that this redistribution of mass can lead to sudden non-local change in momentum near the removed particle and the original two particles. However, we found that this effect is only temporary and is almost unnoticeable. When splitting particles more than once, we track the hierarchy of these parent particles so that their mass can be fully restored.

The pseudo code for our thin sheet preservation is shown in Algorithm 2.

Algorithm 2 PRESERVE THIN FLUID SHEETS

```
C ← Compute Covariance Around p by Eq. 8
RΣR^T ← SVD(C)
P ← Extract Thin Sheet Particles(Σ) by Eq. 11
for all (i, j) ∈ P do
    if a pair (p_i, p_j) satisfies Eq. 12 then
        S ← Insert New Candidate (p_i + p_j)/2
    end if
end for
I_1 ← by Eq. 13 in S
for n = 2, 3, 4, ··· do
    I_n ← search(I_{n−1}) in S as Eq. 14
end for
Insert New Particles I
Collapse Particles that Satisfy Eq. 15
```

5 SOLVER EXTENSIONS

In the following subsections, we describe three extensions to the simulation algorithm discussed so far. These include adaptivity, to reduce the overall number of particles in the simulation, and an anisotropic position correction for even particle distribution.

5.1 Adaptively Sampled Particles

In the FLIP method, the number of liquid particles directly affects the quality of the generated surfaces. Usually, eight particles are initially placed in one simulation grid cell. As the FLIP method cannot capture fluid motion smaller than a single grid cell, we can save large amounts of computational resources by adaptively resampling the bulk volume of the flow with fewer particles while preserving a dense particle sampling near the visible surfaces. This way we can focus the computations on the visible surface detail, and keep the high particle density there, while reducing the number of particles used to sample the bulk volume. Our adaptive sampling model is inspired by the methods of Adams et al. [38] and Hong et al [39], [40], which we have adapted for integration into the FLIP simulation framework. The changes to the previous methods are twofold. First, rather than placing multiple layers and judging particle split by Reynolds number as in [39], [40], we only check whether particles exist close to the free surface for merging and splitting particles. Second, because the FLIP simulation is stable regardless of the space between particles, we simply place particles at random positions without finding nearby free space as done by [38]. In addition, we describe a new simple method to merge particles that can be performed in parallel.

5.1.1 Particle Merge

In the following, we will divide the simulation region into *surface cells*, which are all cells close to the liquid interface, and *deep cells*, which are all remaining ones inside the fluid volume. The surface cells can be easliy found by dilating the air cells a few times. In our model, initial densely placed particles are called "small" particles, and the sparse particles in the bulk volume are labeled as "big" particles. The big particles are generated by merging small particles, up to a certain maximal number. For every particle that exists in the deep water cells, we merge particles based on the following rules:

1) Find particle pairs that are close enough to one another. Typically, closer than the particle radius.
2) If the pair consists of one or two big particles, we compute the sum of all contained small particles. If the number exceeds limited tolerance m, we skip merging for this pair.
3) Otherwise, we generate a big particle from the pair. The radius of this particle is modified such that

$$d_n = n^{\frac{1}{D}} d_0, \tag{16}$$

where d_n denotes a big particle radius that contains n small particles, while d_0 is the particle radius of the initial small particles. D denotes the simulation dimension (D=2 in 2D and 3 in 3D). In our experiments, we found it best to set the maximum number m such that the radius of the biggest particle is the size of simulation grid cell:

$$m = (\frac{\Delta x}{d_0})^D \tag{17}$$

4) The new big particle is placed at the midpoint of the existing pair. Its mass is updated to be the sum of the masses of all contained small particles. The other physical quantities such as velocity are interpolated based on their mass.

We repeat this process until no particles are left to be merged. Because we merge particles in parallel, we need to avoid the duplication of particles in multiple pairs. To ensure this, we only consider pairs between the two closest neighboring particles. If a pair references a particle that is not the closest neighbor, we skip it. This process is illustrated in Figure 8.

5.1.2 Particle Split

When big particles enter surface cells, the merging has to be undone. We split the big particle into the correct number of small particles, which are placed around the source particle at random positions within radius d_n. The mass of the big particle is evenly distributed among the newly created small particles, and their radii are initialized accordingly. All the other physical quantities are copied.

Such a particle split could cause sudden large spring force in a regular SPH simulation. However, as the regular FLIP algorithm does not apply forces based on the particle density this is unproblematic. In the next section, we describe an incremental particle displacement technique to ensure even spacing of the particles. A particle split influences the generated force, but due to its small magnitude we did not observe any instabilities due to the particle splitting.

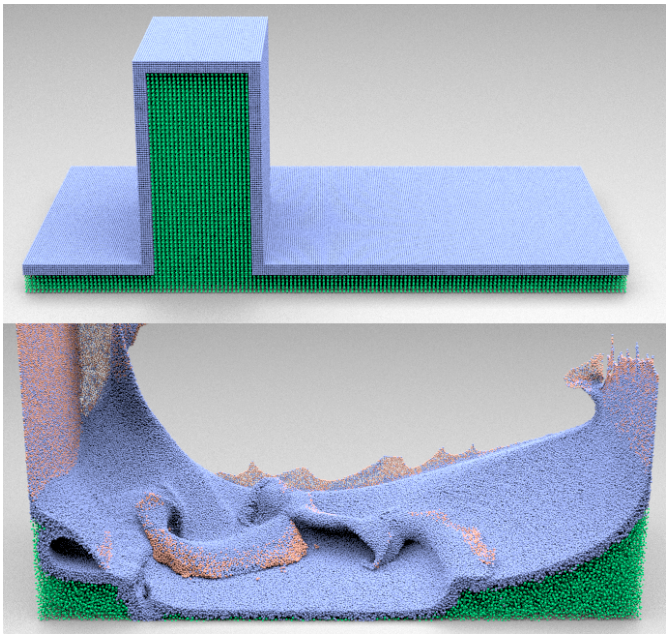


Fig. 7. **Adaptive sampling cutaway view.** Top: Initial placement of particles. Bottom: Simulated particles with described adaptive sampling.

Fig. 8. **Particle merging.** Left: Particle pairs which are closer than the particle radius. Middle: Pairs which references the closest two neighboring particles. Right: After the parallel merging. We repeat these steps until no particles are left to be merged.

5.1.3 Algorithm

Each step of the merging and splitting processes can be done in parallel. Splitting operations can be performed very efficiently since no neighborhood information is required. In our implementation we insert a layer of cells between the surface and the deep fluid layers where no conversions between small and big particles are performed. This prevents excessive merging and splitting operations that could otherwise happen at the interface between deep and surface cells.

We found that adaptively merging particles can effectively reduce the overall number of particles in the simulation, and significantly improves the overall performance. Merging particles in this way can influence the velocities computed during the FLIP simulation for the

Algorithm 3 ADAPTIVELY SAMPLE PARTICLES

DeepCells ← FluidCells − dilate(EmptyCells,twice)
SurfaceCells ← FluidCells − dilate(DeepCells,once)
repeat
  for all particles $(\boldsymbol{p}_i, \boldsymbol{p}_j) \in$ DeepCells **do**
    skip if $||\boldsymbol{p}_i - \boldsymbol{p}_j|| > (d_i + d_j)/2$
    skip if $\arg\min_k ||\boldsymbol{p}_i - \boldsymbol{p}_k|| \neq j$
    skip if $\arg\min_k ||\boldsymbol{p}_j - \boldsymbol{p}_k|| \neq i$
    $n \leftarrow$ Sum of small particles.
    skip if $n > m$
    $\boldsymbol{p}_k \leftarrow$ Generate a new particle.
    $d_k \leftarrow n^{\frac{1}{D}} d_0$
    Remove the particles $\boldsymbol{p}_i$, $\boldsymbol{p}_j$
  end for
until no particles merge
for all merged particle $\boldsymbol{p}_k \in$ SurfaceCells **do**
  for $i = 1 \rightarrow$ number of particles in $\boldsymbol{p}_k$ **do**
    $\boldsymbol{p}_i \leftarrow$ Generate a new particles at random position around $\boldsymbol{p}_k$ within radius d_k.
    $d_i \leftarrow d_0$
  end for
  Remove particle $\boldsymbol{p}_k$
end for

deep cells. However, we have found that this influence is negligible, and barely changes the position of the visible surface. We will demonstrate the improved performance and the effect of the adaptive resampling in the results section below. The effect of the adaptive particle sampling can be seen, e.g., in Figure 2 and Figure 7. The pseudo code for our adaptive sampling algorithm is shown in Algorithm 3.

5.2 Anisotropic Position Correction

An inherent problem of PIC and FLIP is that both methods can lead to uneven spatial particle distributions over time, which in turn can cause "holes" due to missed fluid cells, bumpy surfaces and unreliable adaptive sampling. To prevent this, we slightly move particles along the direction of an SPH-like pressure force, and resample the particle velocities.

If we neglect the current particle distribution, an isotropic version of a position displacement vector $\Delta\boldsymbol{p}$ could be computed as follows:

$$\Delta\boldsymbol{p}_i = -\Delta t \gamma_s d_i \sum_j \frac{\boldsymbol{p}_j - \boldsymbol{p}_i}{||\boldsymbol{p}_j - \boldsymbol{p}_i||} W_{smooth}(\boldsymbol{p}_j - \boldsymbol{p}_i, d_i), \quad (18)$$

where γ_s denotes the stiffness of the displacement. We have used $\gamma_s = 50$. The particle positions are slightly moved to $\boldsymbol{p}_{\text{new}} = \boldsymbol{p} + \Delta\boldsymbol{p}$. Next, we recompute the velocity of the particles which is defined by Equation 1 and the positions and velocities $\boldsymbol{u}$ before applying the displacement. A new particle velocity is computed such that $\boldsymbol{u}_{\text{new}} = \boldsymbol{u}(\boldsymbol{p}_{\text{new}})$ to keep this physical quantity over the domain consistent during the displacement. Note that the mass of a particle is neglected when applying the displacement, since it works purely as a position correction.

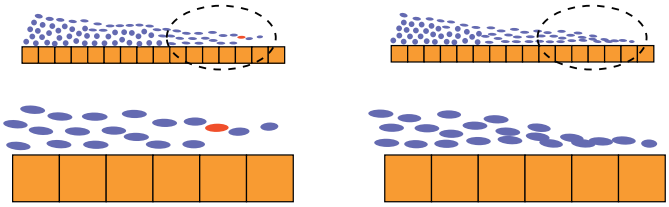


Fig. 9. **Effect of the anisotropic position correction.** 2D comparison of the isotropic (left) and anisotropic displacement correction (right). The anisotropic displacement correction avoids a thickening of thin sheets, and keeps the sharp front intact.

This isotropic version of the displacement was used in Ando and Tsuruno's work [49], but the method can lead to a thickening of thin surfaces, or a smearing out of sharp features. We can alleviate this issue as we have the particle stretch information from the SVD ready from previous steps, and we can take this information into account when applying the position correction. We propose to use the following anisotropic version of the position displacement vector:

$$\Delta\boldsymbol{p}_i = -\Delta t \gamma_s d_i \sum_j \frac{\boldsymbol{r}_{i,j}}{||\boldsymbol{r}_{i,j}||} W_{smooth}(\boldsymbol{r}_{i,j}, d_i) \quad (19)$$

$$\boldsymbol{r}_{i,j} = \frac{1}{k_s} C_i^{-1} (\boldsymbol{p}_j - \boldsymbol{p}_i) \quad (20)$$

where k_s denotes a scaling constant such that $||k_s C_i|| \approx 1$. As before, we constrain the eigenvectors of the anisotropy matrix to lie within a certain range. For the displacement correction we have used $0 < \sigma_n < 1/k_s$. Equation 19 can be symmetrized as:

$$\Delta\boldsymbol{p}_i = -\Delta t \gamma_s d \sum_j \frac{\boldsymbol{r}}{||\boldsymbol{r}||} W_{smooth}(\boldsymbol{r}, d) \quad (21)$$

$$\boldsymbol{r} = \frac{1}{2k_s}(C_i^{-1} + C_j^{-1})(\boldsymbol{p}_j - \boldsymbol{p}_i), \quad d = \frac{1}{2}(d_i + d_j). \quad (22)$$

In this paper we applied this anisotropic displacement vector to maintain a uniform particle distributions. A two-dimensional comparison is shown in Figure 9.

We apply a very weak position displacement and the resampling of velocity to prevent any noticeable disturbances of the simulation. As FLIP simulations tend to slowly lead to uneven particle distributions over time, our method gradually counteracts this problem. An advantage of our approach in comparison to SPH simulations is that it allows us to take large time steps, which could lead to stability problems in SPH. It should be taken into account that the re-computation of the new particle velocities introduces a certain amount of additional diffusion. This can be prevented by changing the viscosity α_ν of the FLIP simulation accordingly. In addition, we found this slightly more viscous behavior to be appropriate for generating liquid animations with

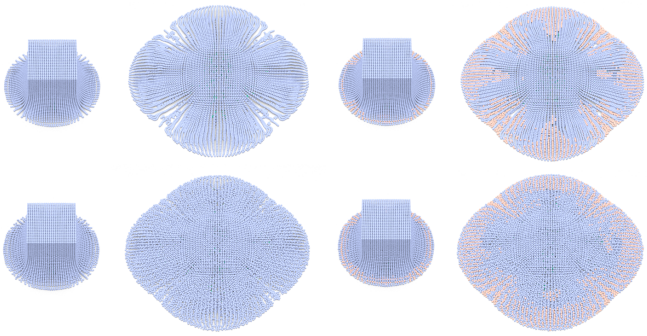


Fig. 10. **Effect of our incremental position correction.** Top: uniformly placed particles. Bottom: same particle placement with corrective position displacement applied. Left column: normal simulation. Right column: simulation with our thin sheet preservation.

thin features. Algorithm 4 shows the pseudo code for our anisotropic position correction computation.

Algorithm 4 ANISOTROPIC POSITION CORRECTION

1: **for all** $i \in P$ **do**
2: $\quad \Delta \boldsymbol{p}_i \leftarrow$ Eq. 21
3: $\quad \boldsymbol{p}_i^{\text{new}} = \boldsymbol{p}_i + \Delta \boldsymbol{p}_i$
4: $\quad \boldsymbol{u}_i^{\text{new}} = \boldsymbol{u}(\boldsymbol{p}_i^{\text{new}})$ using Eq. 1
5: **end for**

Figure 10 shows a three-dimensional comparison with and without the position correction. Without the correction, the particle distribution can develops unevenly. In this scenario, initially randomized particles [11] may alleviate the issue, but this technique works only at the beginning of the simulation; eventually, the distribution will become uneven. In contrast, our method is capable of also fixing this issue during the course of a longer simulation.

As can be seen on the right side of Figure 10, the particle position correction also improves the results of our sheet preservation algorithm. Without the correction, our algorithms is frequently triggered to fill the gaps, but can fail to find a suitable position for newly created particles. In the worst case, this can lead to a sheet breaking unevenly. In combination with the anisotropic position correction, the thin sheet preservation produces the desired result, as can be seen in the bottom-right image of Figure 10.

5.3 Surface Reconstruction

We reuse the SVD information of Equation 10 for evaluating anisotropic kernels for the surface reconstruction, as proposed in [13]. For the simulations of this paper we employed a simple implicit function:

$$\phi_{\text{S}}(\boldsymbol{x}) = \min_i(||C_i^{-1}(\boldsymbol{p}_i - \boldsymbol{x})||), \tag{23}$$

We compute a triangulation of the surface using the Marching Cube algorithm [5]. To make sure all thin

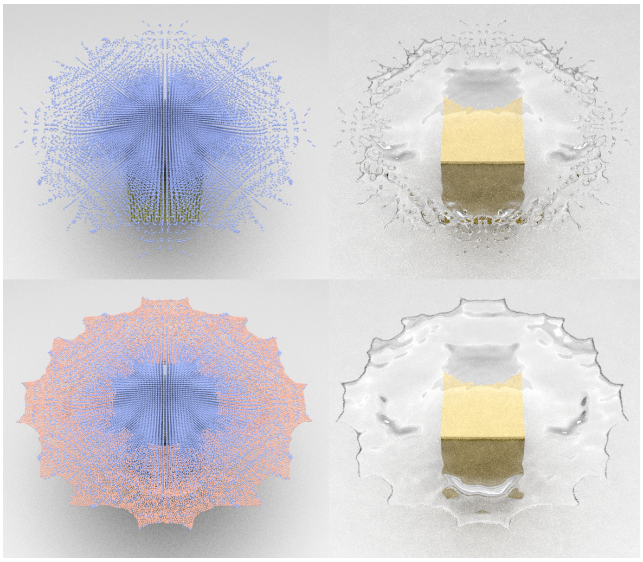


Fig. 11. **Water drop on cubic tower.** Top row: liquid solver only. Bottom row: the same simulation with our sheet preserving algorithm. The left hand side visualizes the particles, while the right hand side shows the transparently rendered mesh.

sheets were captured, we set the minimal eigenvalue of C_i be to on the order of a single cell. In addition, we applied a straightforward mesh-based smoothing to reduce any remaining bumps in the resulting surface.

6 RESULTS

All of the following simulation results were run on a Core i7 860 2.8 GHz PC using a resolution of 100^3, except for the simulation shown in Figure 19 and 18, which was simulated at 150^3 on a Core i7-2600k 3.4 GHz PC. Every liquid solver step is done in a parallel manner. This includes mapping between the grid and a particle, the adaptive sampling, solving for pressure, and applying the anisotropic position correction. The thin feature preserving algorithm is also parallelized, except for particle insertion and collapse. In our implementation we gained significant acceleration by employing OpenMP directives. The cost of computing the list of neighbors was less than 0.5% of the simulation time, so we reconstruct the list of neighbors whenever it is necessary. The simulation itself requires between 5 and 40 seconds per time step for the 100^3 examples shown here. For the time step size, we chose $\Delta t = 0.6 \times 10^{-2}$. The surface mesh creation took up to 4 seconds for a resolution of 200^3 and the renderings with transparency took approximately two minutes per frame. Transparent renderings were done with the Pixie and Yafaray renderers, while we performed the point and opaque renderings using PBRT.

Figure 11 shows an example of a water drop hitting a cubic tower, resulting in a thin sheet expanding in a circular manner. Without our thin sheet preservation, the liquid sheet quickly ruptures as it expands. With

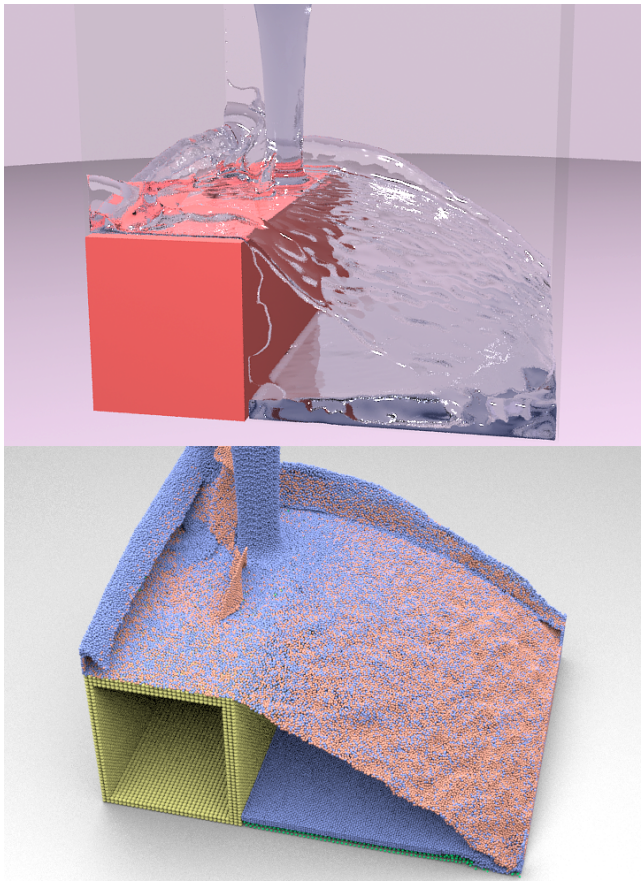


Fig. 12. **Thin sheet at ledge.** Top: Liquid is poured onto an obstacle and flows over the ledge, forming a thin sheet and hitting the pool below. Bottom: Particle cutaway view.

our approach, we successfully maintain a continuous liquid sheet. Our algorithm successfully identifies the regions where the sheet would break up, and inserts new particles in the appropriate regions.

Figure 12 shows an example of water being poured onto an obstacle, and forming a thin sheet while flowing off the ledge. Notice that the stream of liquid flowing down hits the pool without rupturing. Each time step took approximately 5 to 40 seconds to compute. The number of liquid particles was $400k$ at the frame shown in Figure 12. Note that although we limit the number times that a particle can split, we could not completely drain the liquid on the box for this example, and a thin layer of liquid remained on top of the box. If necessary, the particles on top could be identified and removed as in [49] However, we deliberately choose not to do this in order to avoid prevent the introduction of potential visual artifacts.

Figure 13 shows a simulation of a water drop hitting a pool of liquid, comparing a simulation with and one without our sheet preserving algorithm. With our algorithm, we are able to successfully capture the splash around the drop without unwanted holes in the liquid sheet. On the other hand, in the bottom image, the sheet of liquid dissolves into individual particles. For

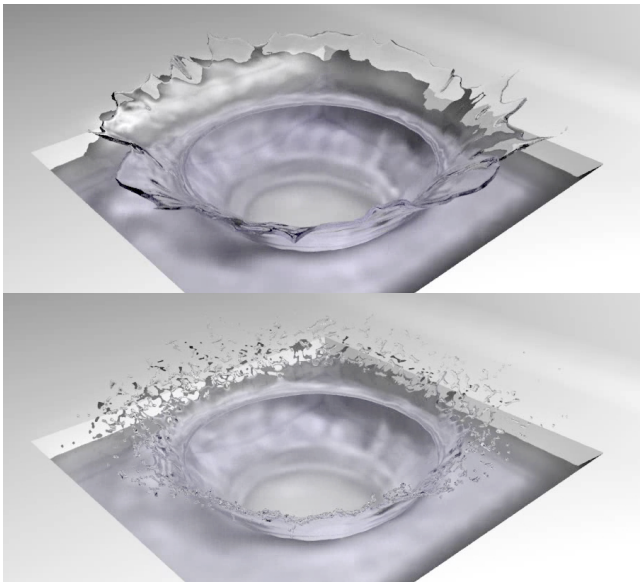


Fig. 13. **Water drop in a container.** Top: With our sheet preserving algorithm. Bottom: Without the sheet preserving algorithm. Notice that without our our method the thin sheets can easily rupture.

this simulation, a single time step took 20 seconds, and $400k$ particles were used on average.

To demonstrate that the adaptive sampling does not introduce simulation artifacts, a comparison between a simulation using this approach, and a regular one can be seen in Figure 18. We observed that adaptively sampled particles tend to lose some of the active motions of the surfaces, resulting in a slightly faster settling of the liquid. This is most likely caused by the adaptive particle splitting and merging processes, which leads to an extra diffusion of the fluid velocity. We plan to evaluate more accurate particle-based interpolation schemes (e.g. radial basis functions) to alleviate this issue as future work.

A test case often used for evaluating surface tracking algorithms is the Enright deformation test [50]. In this test, a solid sphere is deformed according to an artificial flow field, which is then reversed to restore the sphere to its original shape. Figure 17 shows how our approach performs for this problem. The thin sheet, which can quickly break up using traditional level set methods, is well preserved due to the extra particle insertion, shown in pink. These particles are then removed as the deformation returns the sphere to its original form.

Timings for each step of our fluid solver during the course of a simulation are shown in Figure 16. The most time-consuming part of our simulator changes depending on the situation. When the simulation contains large amounts of thin regions, the sheet preserving algorithm can consume most of the simulation time. However, on average, the FLIP solver was the most time consuming part. The two left-most graphs, a and b, compare the performance of a simulation using our adaptive particle re-sampling strategy, and one without.

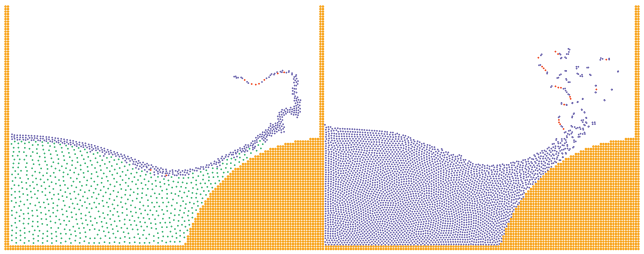


Fig. 14. **Comparison with SPH.** The left image shows a FLIP based solver, while the right one shows an SPH based simulation. Both use our thin sheet preservation. Notice that the SPH simulation fails to accurately resolve and preserve the thin sheets.

Name	Description	Value
α_u	Velocity radius scale	1.0
α_ρ	Density radius scale	4.0
α_L	Liquid levelset rate	0.2
α_ν	FLIP viscosity	0.95
$\alpha_{\partial\Omega}$	Surface particle density rate	0.7
β_{thin}	Thin sheets rate	0.2
β_{min}	Insertion minimum distance	0.8
β_{max}	Insertion maximum distance	3.5
$\beta_{\text{max}\rho}$	Thin particle maximum density	0.2
β_{dist}	Thin particle minimum distance	0.2
γ_s	Displacement stiffness	50
Δt	Time step size	0.6×10^{-2}
N_0	Number of particles in a cell	8

Fig. 15. **Parameters.** We used these specific parameters to generate example animations shown in this paper.

The position correction and the density computation steps require particle neighborhoods, and thus can take significant amounts of time to compute, as can be seen in Figure 16b. However, thanks to the overall reduction of the number particles with adaptive sampling, we can minimize this cost considerably, as can be seen in Figure 16c.

7 Discussion

As our method is based on particles, it can also be applied to SPH simulations. Figure 14 shows a comparison of SPH and our FLIP-based fluid solver coupled with our sheet preserving algorithm. Although the method works in principle, we found the SPH simulations to have particle motions that were too violent to accurately resolve sheets with our approach.

As can be seen from the figure, the splashy behavior of the SPH makes it hard to detect unique thin fluid features to be preserved. Note that we did not employ surface tension forces in the SPH simulation, which could be used to force the liquid into temporally more coherent structures.

Our position correction algorithm can be considered as a kind of re-meshing process, and thus must be applied to positions. If were to apply the correction directly to the velocities as a force this could easily lead to unrealistic behavior, since such forces do not exist in the governing physics.

Our algorithm has several parameters. So far, we have introduced 13 user-adjustable parameters $\alpha_u \sim \gamma_s$, N_0 and Δt, but more are possible in an actual implementation. In detail, our fluid simulator has 8 parameters and the sheet preserving algorithm has 5 parameters. Fortunately, we found that these parameters are not sensitive in terms of stability, and we were able to use constant values for all of them for the simulations shown in this work. Δt can be heuristically changed but it does not have to be overly small to keep the simulation stable. We show a list of the parameters that were used in Table 15.

With adaptive sampling, our thin sheet preservation sometimes finds small "gaps" between small particles and merged particles. This is because our sheet preserving algorithm "misinterprets" thin layers of small particles as a thin sheet, and tries to fill holes between the surface particles and the particles in the bulk volume. This problem could be fixed by a more thorough check of the particle types, but we did not observe any side effects due to this misinterpretation.

There are some specific types of shapes that our algorithm does not handle well. For instance, our method can have problems with separating "T-junctions" or "V" shaped fluid sheets. For these shapes, the SVD interprets the articulated joints as a thick area, and prevents particle splitting there. We believe this issue can be alleviated by propagating "thin particles" labels among particles.

Note that in some cases the fluid sheet seems to expand without limit as it stretches, but eventually breaks into individual drops because we skip splitting when the density is too low. Moreover, the anisotropic kernels could also be used to search nearby particles when splitting particles in our sheet preserving algorithm. An anisotropic splitting could fill ruptures more robustly than our isotropic method does. We plan to investigate the effects of such a modification in more detail in the future.

For our adaptive sampling approach, the largest size of a merged particle is currently limited by the size of the grid cell. It would be possible to allow particles to grow to larger sizes. However, because the underlying grid is uniform, we cannot fully benefit from the sampled particles when running the actual FLIP solver. As future work, we would like to also adaptively re-mesh the Eulerian grid in accordance with the particles in order to fully benefit from the adaptivity, as was done by Hong et al. [40] and Sin et al. [10].

Also note that although we applied our algorithm only to liquids in this work, it would also be applicable to particle based representations of other phenomena, for example viscoelastic materials or wispy smoke, which we leave as future work.

8 Conclusion

In this paper we have demonstrated a particle-based algorithm that preserves thin fluid sheets by carefully re-sampling particles. As an underlying fluid solver, we employed the FLIP method extended with an incremental anisotropic position correction to ensure even particle distribution. To reduce the cost of neighborhood computations, we adaptively sampled particles in the deep water while retaining dense particles near surfaces. To prevent fluid sheets from breaking, we identified thin fluid regions by computing the stretch of the neighboring particle distributions. Within the critical regions, new particles were carefully introduced to avoid disturbing the underlying flow when filling sparsely sampled areas. We have shown a variety of different simulations to demonstrate that our approach can efficiently handle a variety of liquid phenomena with thin features.

Acknowledgements

This work was supported by the Japan Society for the Promotion of Science (JSPS) and the Grants-in-Aid for Scientific Research (23611021).

References

[1] C. Wojtan and G. Turk, "Fast viscoelastic behavior with thin features," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–8, 2008.

[2] M. Müller, "Fast and robust tracking of fluid surfaces," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '09. ACM, 2009, pp. 237–245.

[3] T. Brochu, C. Batty, and R. Bridson, "Matching fluid simulation elements to surface geometry and topology," *ACM Trans. Graph.*, vol. 29, pp. 47:1–47:9, July 2010.

[4] C. Wojtan, N. Thürey, M. Gross, and G. Turk, "Physics-inspired topology changes for thin fluid features," *ACM Trans. Graph.*, vol. 29, pp. 50:1–50:8, July 2010.

[5] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, pp. 163–169, August 1987.

[6] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ser. SCA '03. Eurographics Association, 2003, pp. 154–159.

[7] P. Goswami, P. Schlegel, B. Solenthaler, and R. Pajarola, "Interactive sph simulation and rendering on the gpu," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '10. Eurographics Association, 2010, pp. 55–64.

[8] T. Harada, S. Koshizuka, and Y. Kawaguchi, "Smoothed particle hydrodynamics on gpus," in *Computer Graphics International*, 2007, pp. 63–70.

[9] H. Yan, Z. Wang, J. He, X. Chen, C. Wang, and Q. Peng, "Real-time fluid simulation with adaptive sph," *Computer Animation and Virtual Worlds*, vol. 20, no. 2-3, pp. 417–426, 2009.

[10] F. Sin, A. W. Bargteil, and J. K. Hodgins, "A point-based method for animating incompressible flow," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '09. ACM, 2009, pp. 247–255.

[11] Y. Zhu and R. Bridson, "Animating sand as a fluid," *ACM Trans. Graph.*, vol. 24, pp. 965–972, July 2005.

[12] F. Losasso, J. Talton, N. Kwatra, and R. Fedkiw, "Two-way coupled sph and particle level set fluid simulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 797–804, July 2008.

[13] J. Yu and G. Turk, "Reconstructing surfaces of particle-based fluids using anisotropic kernels," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '10. Eurographics Association, 2010, pp. 217–225.

[14] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations," *J. Comput. Phys.*, vol. 79, no. 1, pp. 12–49, 1988.

[15] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell, "A hybrid particle level set method for improved interface capturing," *J. Comput. Phys*, vol. 183, pp. 83–116, 2002.

[16] Z. Wang, J. Yang, and F. Stern, "An improved particle correction procedure for the particle level set method," *J. Comput. Phys.*, vol. 228, pp. 5819–5837, September 2009.

[17] V. Mihalef, D. Metaxas, and M. Sussman, "Textured liquids based on the marker level set," *Computer Graphics Forum*, vol. 26, no. 3, pp. 457–466, 2007.

[18] A. W. Bargteil, T. G. Goktekin, J. F. O'brien, and J. A. Strain, "A semi-lagrangian contouring method for fluid simulation," *ACM Trans. Graph.*, vol. 25, pp. 19–38, January 2006.

[19] N. Heo and H.-S. Ko, "Detail-preserving fully-eulerian interface tracking framework," in *ACM SIGGRAPH Asia 2010 papers*, ser. SIGGRAPH ASIA '10. ACM, 2010, pp. 176:1–176:8.

[20] N. Foster and R. Fedkiw, "Practical animation of liquids," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '01. ACM, 2001, pp. 23–30.

[21] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw, "Coupling water and smoke to thin deformable and rigid shells," *ACM Trans. Graph.*, vol. 24, pp. 973–981, July 2005.

[22] J. Kim, D. Cha, B. Chang, B. Koo, and I. Ihm, "Practical animation of turbulent splashing water," in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ser. SCA '06. Eurographics Association, 2006, pp. 335–344.

[23] N. Chentanez, B. E. Feldman, F. Labelle, J. F. O'Brien, and J. R. Shewchuk, "Liquid simulation on lattice-based tetrahedral meshes," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ser. SCA '07. Eurographics Association, 2007, pp. 219–228.

[24] C. W. Hirt and B. D. Nichols, "Volume of fluid vof method for the dynamics of free boundaries," *Journal of Computational Physics*, vol. 39, pp. 201–225, January 1981.

[25] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988.

[26] J. Glimm, J. W. Grove, X. L. Li, K.-m. Shyue, Y. Zeng, and Q. Zhang, "Three-dimensional front tracking," *SIAM J. Sci. Comput.*, vol. 19, no. 3, pp. 703–727, 1998.

[27] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y. Jan, "A front-tracking method for the computations of multiphase flow," *Journal of Computational Physics*, vol. 169, no. 2, pp. 708–759, 2001.

[28] T. Brochu and R. Bridson, "Robust topological operations for dynamic explicit surfaces," *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 2472–2493, 2009.

[29] C. Wojtan, N. Thürey, M. Gross, and G. Turk, "Deforming meshes that split and merge," in *ACM SIGGRAPH 2009 papers*, ser. SIGGRAPH '09. ACM, 2009, pp. 76:1–76:10.

[30] M. Becker and M. Teschner, "Weakly compressible sph for free surface flows," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ser. SCA '07. Eurographics Association, 2007, pp. 209–217.

[31] B. Solenthaler and R. Pajarola, "Predictive-corrective incompressible sph," *ACM Trans. Graph.*, vol. 28, pp. 40:1–40:6, July 2009.

[32] S. Koshizuka, H. Tamako, and Y. Oka, "A particle method for incompressible viscous flow with fluid fragmentation," *Comput. Fluid Dynamics J.*, vol. 29(4), 1996.

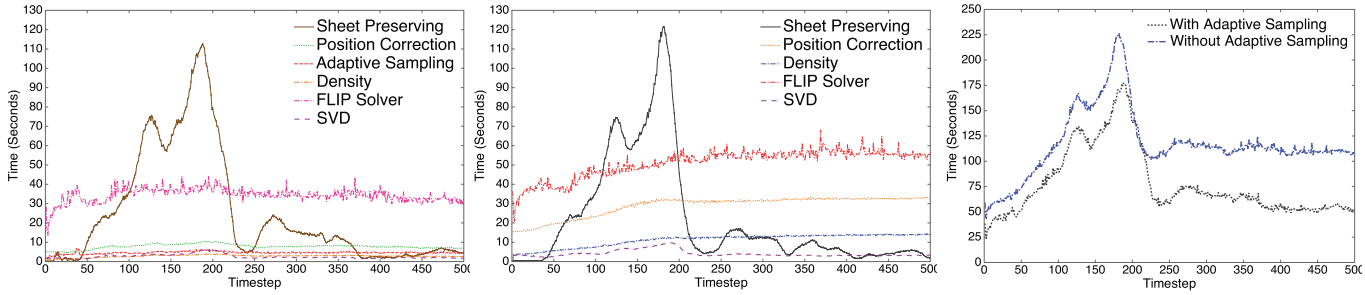
[33] S. Premžoe, T. Tasdizen, J. Bigler, A. Lefohn, and R. T. Whitaker, "Particle-based simulation of fluids," *Computer Graphics Forum*, vol. 22, no. 3, pp. 401–410, 2003.

[34] M. Pauly, R. Keiser, B. Adams, P. Dutré, M. Gross, and L. J. Guibas, "Meshless animation of fracturing solids," *ACM Trans. Graph.*, vol. 24, pp. 957–964, July 2005.

[35] R. Keiser, B. Adams, D. Gasser, P. Bazzi, P. Dutre, and M. Gross, "A unified lagrangian approach to solid-fluid animation," *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics*, vol. 0, pp. 125–148, 2005.

[36] D. Gerszewski, H. Bhattacharya, and A. W. Bargteil, "A point-based method for animating elastoplastic solids," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '09. ACM, 2009, pp. 133–138.

[37] J. F. Blinn, "A generalization of algebraic surface drawing," *ACM Trans. Graph.*, vol. 1, pp. 235–256, July 1982.



(a) With Adaptive Sampling (b) Without Adaptive Sampling (c) Overall Simulation Time

Fig. 16. **Timing results for the dam breaking setup of Figure 19 at resolution** 150^3. At highest load, the sheet preservation algorithm can surpass the core FLIP solver, which takes the most time on average. Adaptive sampling significantly reduces the cost of density and position correction computation, as can be seen in (a) and (b), and accelerates the overall algorithm by a factor of up to two (c).

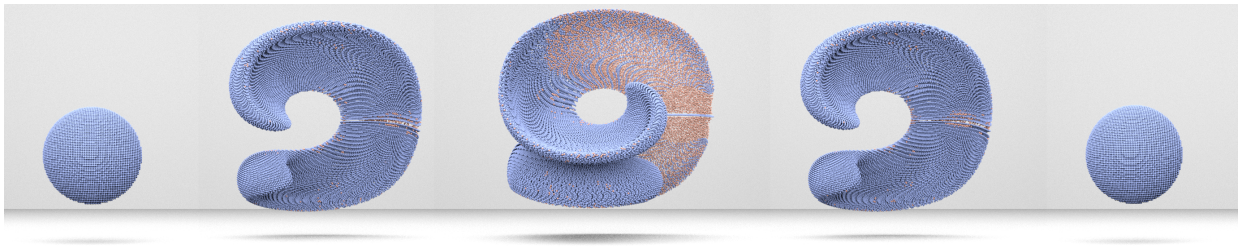


Fig. 17. **Enright deformation test.** Our particle-based method can tolerate the thin features of the Enright test by particles that inserted at the critical points (shown in pink) and collapsed in densely regions.

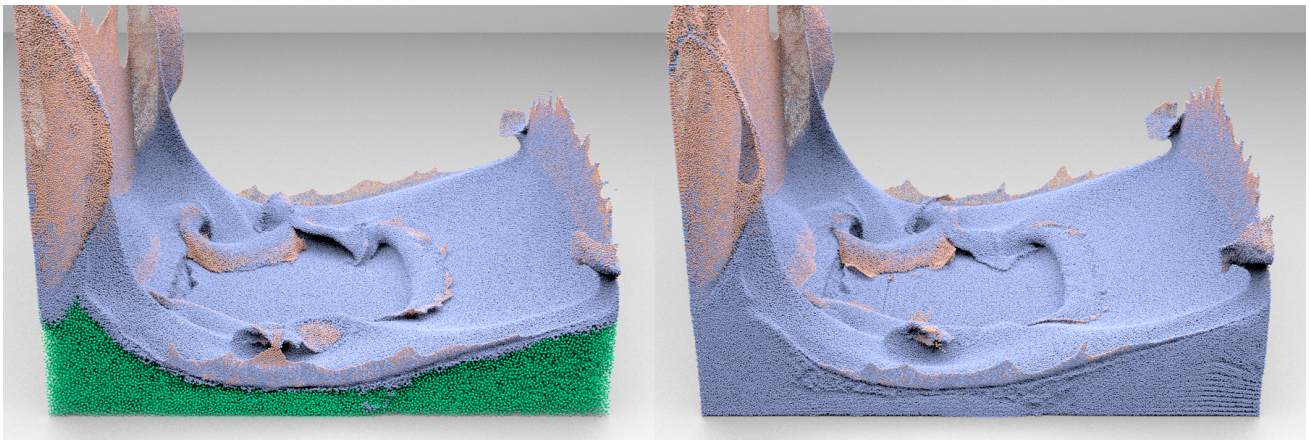


Fig. 18. **Comparison between adaptive re-sampling and regular sampling.** The left side shows our adaptively sampled simulation, while right side shows a regularly sampled simulation. Notice that both exhibit similar motion.

[38] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas, "Adaptively sampled particle fluids," *ACM Trans. Graph.*, vol. 26, July 2007.
[39] W. Hong, D. H. House, and J. Keyser, "Adaptive particles for incompressible fluid simulation," *Vis. Comput.*, vol. 24, pp. 535–543, July 2008.
[40] ——, "An adaptive sampling approach to incompressible particle-based fluid." in *TPCG*, W. Tang and J. P. Collomosse, Eds. Eurographics Association, 2009, pp. 69–76.
[41] K. Raveendran, C. Wojtan, and G. Turk, "Hybrid smoothed particle hydrodynamics," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '11. New York, NY, USA: ACM, 2011, pp. 33–42.
[42] F. H. Harlow, "The particle-in-cell computing method for fluid dynamics," *Methods Comput. Phys*, no. 3, pp. 319–343, 1964.
[43] C. Batty and R. Bridson, "Accurate viscous free surfaces for buckling, coiling, and rotating liquids," in *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation*, July 2008, pp. 219–228.
[44] J. Brackbill and H. Ruppel, "Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions," *Journal of Computational Physics*, vol. 65, no. 2, pp. 314 – 343, 1986.
[45] A. Robinson-Mosher, R. E. English, and R. Fedkiw, "Accurate tangential velocities for solid fluid coupling," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '09. ACM, 2009, pp. 227–236.
[46] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw, "Directable photorealistic liquids," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ser. SCA '04. Eurographics Association, 2004, pp. 193–202.

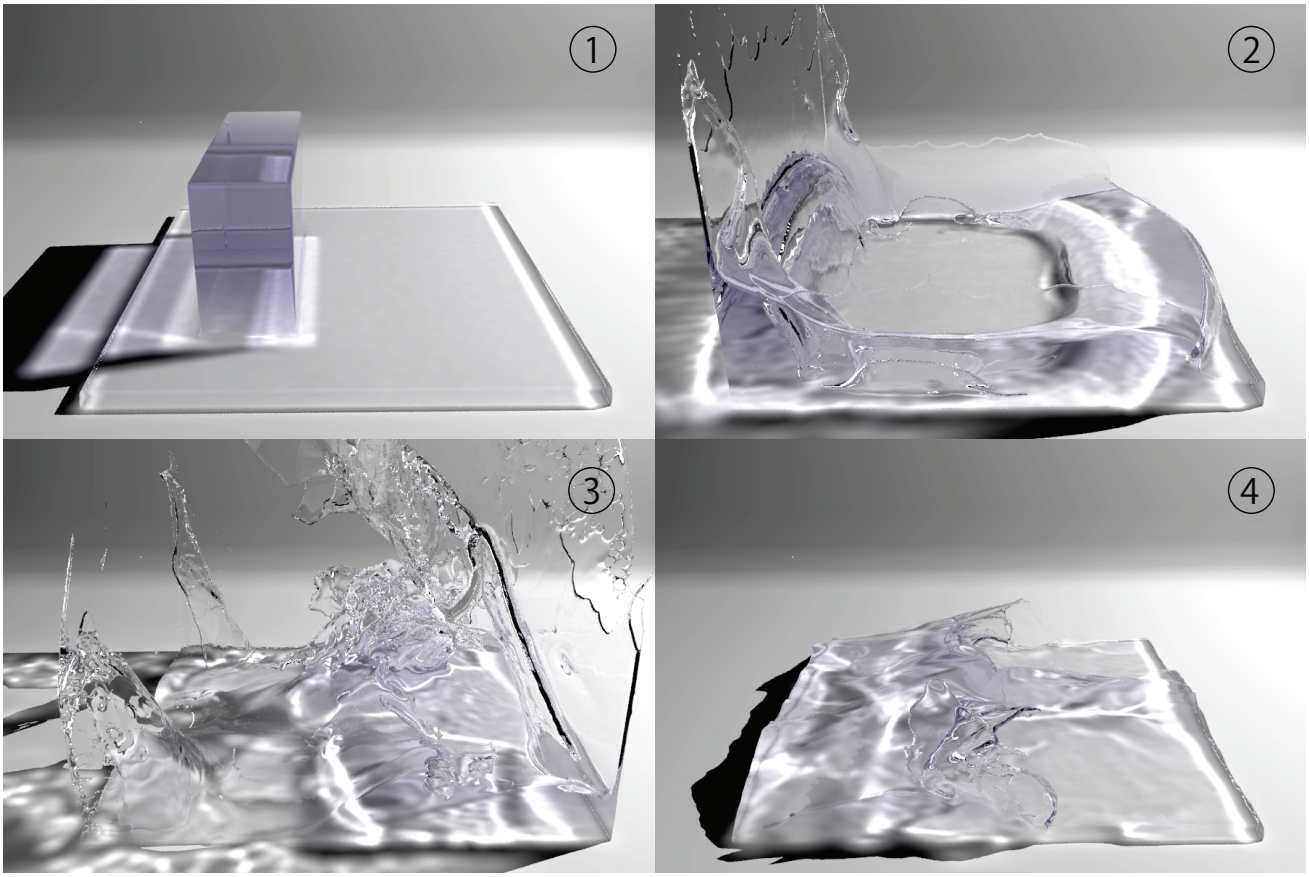


Fig. 19. **Dam breaking setup.** Four images from a dam breaking simulation employing our particle-based thin sheet preservation algorithm.

[47] F. Gibou, R. P. Fedkiw, L.-T. Cheng, and M. Kang, "A second-order-accurate symmetric discretization of the poisson equation on irregular domains," *J. Comput. Phys.*, vol. 176, pp. 205–227, February 2002.
[48] C. Batty, F. Bertails, and R. Bridson, "A fast variational framework for accurate solid-fluid coupling," *ACM Trans. Graph.*, vol. 26, July 2007.
[49] R. Ando and R. Tsuruno, "A particle-based method for preserving fluid sheets," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '11. ACM, 2011, pp. 7–16.
[50] D. P. Enright, "Use of the particle level set method for enhanced resolution of free surface flows," Ph.D. dissertation, 2002, aAI3067855.