Université de Montréal

Hybrid and Data-Driven Methods for Efficient and Realistic Particle-Based Liquid Simulations

par

Bruno Roy

Département d'informatique et de recherche opérationelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures et postdoctorales
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

Décembre 2021

Hybrid and Data-Driven Methods for Efficient and Realistic Particle-Based Liquid Simulations

présentée par

Bruno Roy

Thèse acceptée le :
30 novembre 2021

Sommaire

L'approximation de phénomènes physiques, tels qu'une simulation de liquides en informatique graphique, requiert l'utilisation de méthodes complexes nécessitant des temps de calcul et une quantité de mémoire importants. Malgré les avancées récentes dans ce domaine, l'écart en réalisme entre un liquide simulé et la réalité demeure encore aujourd'hui considérable. Cet écart nous séparant du réalisme souhaité nécessite des modèles numériques de simulation dont la complexité ne cesse de croître. L'objectif ultime est de permettre à l'utilisateur de manipuler ces modèles de simulation de liquides sans la nécessité d'avoir une connaissance accrue de la physique requise pour atteindre un niveau de réalisme acceptable et ce, en temps réel. Plusieurs approches ont été revisitées dans les dernières années afin de simplifier ces modèles ou dans le but de les rendre plus facilement paramétrables. Cette thèse par articles encadre bien les trois projets constituant nos contributions dans le but d'améliorer et de faciliter la génération de simulations de liquides en informatique graphique.

Tout d'abord, nous introduisons une approche hybride permettant de traiter séparément le volume de liquide non-apparent (i.e., en profondeur) et une couche de particules en surface par la méthode de calcul *Smoothed Particle Hydrodynamics* (SPH). Nous revisitons l'approche par bandes de particules, mais cette fois nouvellement appliquée à la méthode SPH qui offre un niveau de réalisme supérieur. Comme deuxième projet, nous proposons une approche permettant d'améliorer le niveau de détail des éclaboussures de liquides. En suréchantillonnant une simulation de liquides existante, notre approche est capable de générer des détails réalistes d'éclaboussures grâce à la dynamique de balistique. En complément, nous proposons une méthode de simulation par vagues permettant de reproduire les interactions entre les éclaboussures générées et les portions quasi-statiques de la simulation existante. Finalement, le troisième projet introduit une approche permettant de rehausser la résolution apparente d'un liquide par l'apprentissage automatique. Nous proposons une architecture d'apprentissage inspirée des flux optiques dont l'objectif est de générer une correspondance entre le déplacement des particules de simulations de liquides à différentes résolutions (i.e., basses et hautes résolutions). Notre modèle d'apprentissage permet d'encoder des caractéristiques de hautes résolutions à l'aide de déformations pré-calculées entre deux liquides à différentes résolutions et d'opérations de convolution basées sur le voisinage des particules.

Mots clés: Liquide par particules, modèle de simulation hybride, bande de particules, éclaboussures de liquide, suréchantillonnage, simulation par vagues, apprentissage automatique, apprentissage profond, rehausse de détails, déformations inter-résolution, convolution par voisinage.

Summary

The approximation of natural phenomena such as liquid simulations in computer graphics requires complex methods that are computationally expensive. Despite recent advances in this field, the gap in realism between a simulated liquid and reality remains considerable. This disparity that separates us from the desired realism requires numerical models whose complexity continues to grow. The ultimate goal is to provide users the capacity and tools to manipulate these liquid simulation models to obtain acceptable realism. In the last decade, several approaches have been revisited to simplify and to allow more flexible models. In this dissertation by articles, we present three projects whose contributions support the improvement and flexibility of generating liquid simulations for computer graphics.

First, we introduce a hybrid approach allowing us to separately process the volume of non-apparent liquid (i.e., in-depth) and a band of surface particles using the Smoothed Particle Hydrodynamics (SPH) method. We revisit the particle band approach, but this time newly applied to the SPH method, which offers a higher level of realism. Then, as a second project, we propose an approach to improve the level of detail of splashing liquids. By upsampling an existing liquid simulation, our approach is capable of generating realistic splash details through ballistic dynamics. In addition, we propose a wave simulation method to reproduce the interactions between the generated splashes and the quasi-static portions of the existing liquid simulation. Finally, the third project introduces an approach to enhance the apparent resolution of liquids through machine learning. We propose a learning architecture inspired by optical flows by which we generate a correspondence between the displacement of the particles of liquid simulations at different resolutions (i.e., low and high resolutions). Our training model allows high-resolution features to be encoded using pre-computed deformations between two liquids at different resolutions and convolution operations based on the neighborhood of the particles.

Keywords: Particle-based liquid, hybrid simulation model, particle band, liquid splash, upsampling, wave simulation, machine learning, deep learning, up-res details, inter-resolution deformations, neighborhood convolutions.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

2/3D	Two/Three dimensions	LHS	Left-hand side
ADAM	Adaptive moment estimation	LSTM	Long short-term memory
CFD	Computational fluid dynamics	MAC	Marker-and-cell
CFL	Courant–Friedrichs–Lewy	MIC	Modified Incomplete Cholesky
CG	Conjugate Gradient	MLP	Multilayer perceptron
CNN	Convolutional neural network	NN	Neural network
CPU	Central processing unit	PBD	Position-based dynamics
DFSPH	Divergence-free SPH	PBF	Position-based fluids
DNN	Deep neural network	PCISPH	Predictive–corrective ISPH
EPE	estimated position error	PDE	Partial derivative equations
FFNet	Fluid flow network	PIC	Particle-in-cell
FLIP	Fluid implicit particle	PLS	Particle level set
FlOF	Fluid optical flow	RAM	Random-access memory
GAN	Generative adversarial network	ReLU	Rectified linear unit
GPU	Graphics processing unit	RHS	Right-hand side
HDK	Houdini development kit	SDF	Signed distance function
HSV	Hue saturation value	SPH	Smoothed particle hydrodynamics
ICP	Iterative closest point	SWE	Shallow water equations
IISPH	Implicit Incompressible SPH	UpFlOF	Up-resed FlOF
ISPH	Incompressible SPH	VFX	Visual effects

Acknowledgements

Tout d'abord, je souhaite remercier mon épouse, Isabelle, pour son support inconditionnel tout au long de ce long parcours. Elle est à l'essence même de la raison qui m'a permis de persévérer pendant les longues soirées et fins de semaine des dernières années.

Naturellement, je tiens également à remercier mon directeur de thèse, Pierre, pour m'avoir supporté et prêté main-forte pendant mon cheminement au doctorat. Par le fait même, je remercie mon codirecteur, Eric, pour son assistance soutenue et inégalée. Je profite aussi de l'occasion pour remercier les membres de mon jury d'évaluation pour leurs commentaires, suggestions et intérêts sur les contributions présentées dans ma thèse.

Finalement, je salue le soutien de mes amis et ma famille qui ont patiemment enduré mes monologues sur les méthodes numériques, l'apprentissage automatique et les simulations physiques en informatique graphique.

Chapter 1

Introduction

The requirements to achieve the desired level of realism are constantly increasing in the application areas of media and entertainment. This is especially true for physically-based models such as fluid simulations used in movies and video games. These models require a level of complexity that has a direct impact on the interactivity and precision throughout production. The granularity of the discretization of these physical models strongly influences the precision with which details are captured, especially for those details at small scale. In other words, the realism and accuracy of the simulation are closely bound to the resolution of the discretization. In this thesis, we present three contributions to overcome, in different ways, the resolution limitations associated with the discretization of numerical simulations of liquids.

Our first contribution [33] aims to reduce the computation times of traditional Smoothed Particle Hydrodynamics (*SPH*) methods by coupling a band of particles at the surface to a coarse Eulerian grid underneath the surface, dividing respectively fine details from the rest of the simulated liquid. The work of Chentanez et al. [5] on a hybrid grid-*SPH* method is the most similar to ours in spirit. They manage to compute their model in real time but at the cost of some inherent physical differences caused by coupling Shallow Water Equations (*SWE*) on a 3D grid with *SPH* particles making the stability of the simulation highly sensitive to an adequate parameterization. In comparison, our Eulerian-based approach with fictitious particles allows us to retain the realism and stability provided by a state-of-the-art *SPH* method. Despite similarities of our approach with the narrow-band Fluid Implicit Particle (*FLIP*) method [11], we propose in comparison a non-homogeneous way (as presented by Desbrun [7]) to deal with density fluctuation and to constrain the particle band around the fluid interface. Our approach is very effective with large bodies of liquid while preserving *SPH* small-scale surface tension details where it matters.

For our third and last contribution [34], we propose an up-resing approach leveraging deep learning to express scene flows as Lagrangian motions for particle-based liquids. Although a few methods have been proposed recently to increase the apparent resolution of fluid flows, our approach is the first to tackle the up-resing problem of liquid simulations from a Lagrangian perspective. In contrast with Xie et al. [43], our up-resing approach is applied directly to the particles of liquid simulations. By decoupling the up-resing part from the advection step, our approach offers a more efficient way to encode and generalize local high-resolution features. Also, as we infer Lagrangian displacements, we propose modified convolution layers inspired from Liu et al. [23], allowing us to use the neighborhood contributions as with *SPH* models. In comparison with *FlowNet3D*, our network combines a pair of highly deformable particle-based liquids by leveraging feature similarities to encode the motion between them.

The thesis is organized as follows: We first provide an introduction to the fundamentals of fluid simulations in computer graphics (Sections 1.1 to 1.4) to properly understand the contributions covered by our projects. The following three chapters respectively present the research projects covering our contributions: a narrow band for *SPH* (Chapter 2), a post-processing upsampling for splashing particle-based liquids (Chapter 3), and a deep-learning approach for up-resing liquids (Chapter 4). Each of these research contributions is supported by a publication. Finally, we conclude by discussing promising avenues for future work (Chapter 5) for each of our contributions and liquid simulations in general.

1.1. Discretized Liquids for Media and Entertainment

The movement of a fluid (e.g., liquid, smoke, sand, snow, and so on) remains difficult to describe and compute effectively. In computer graphics, as we mainly focus on the visual aspect of the resulting animation, we are inclined to approximate the behavior and interactions of fluids while maintaining realism. In the following sections, we briefly summarize the fundamentals of numerical simulations and flow representations for liquids.

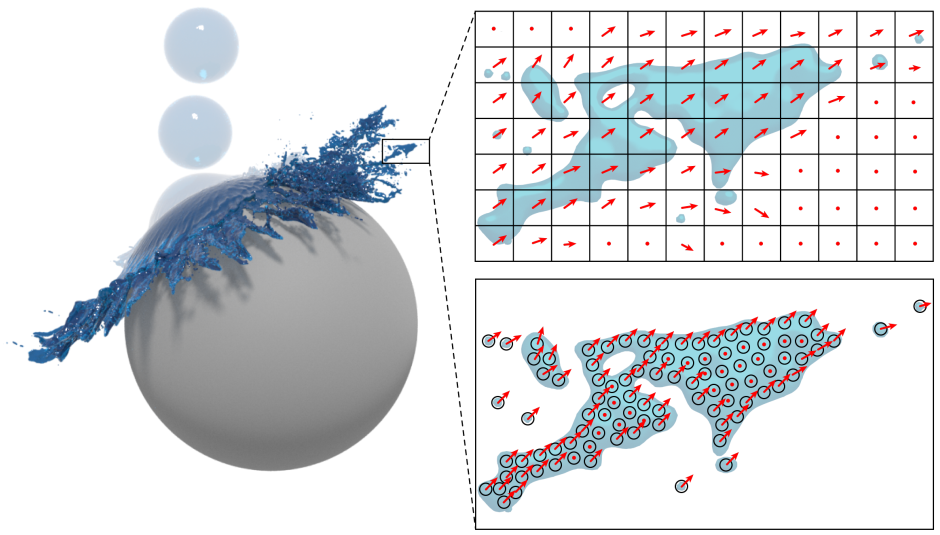


Figure 1.1. Comparison between the Lagrangian (bottom) and Eulerian (top) viewpoints of an impact for a simulated liquid drop (left).

1.1.1. Liquid Motion with Numerical Simulations

In computer graphics, the behavior of most fluids of interest is described by the *incompressible Navier-Stokes equations*:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{\nabla p}{\rho} = \vec{F}_{\text{ext}} + \nu \nabla \cdot \nabla \vec{u}, \quad (1.1)$$

$$\nabla \cdot \vec{u} = 0, \quad (1.2)$$

defined within a constrained simulation domain $\Omega \subset \mathbb{R}^d$ where d is the number of dimensions (usually two or three). Eq. 1.1 is called the *momentum equation* and expresses the motion of the fluid mass and density. This equation is an adapted form of Newton's law of motion $\vec{F} = m\vec{a}$ to fluid mechanics. With this formulation, each portion of fluid has a mass m, a volume V, and a velocity $\vec{u}$. The force exerted on each of these portions of fluid is defined by a material derivative D as follows:

$$\vec{F} = m\frac{D\vec{u}}{Dt}, \quad (1.3)$$

where Eq. 1.3 describes a rate of change of a physical quantity (i.e., density, temperature, etc.) of a material measured at time t. The material derivative can also serve as a way of describing the flow with the appropriate viewpoint (more on that in Section 1.1.2). The motion of these physical quantities q over time and within a certain domain Ω is commonly called *advection*.

$$\begin{aligned} \frac{Dq}{Dt} &= 0, & (1.4) \\ \frac{\partial q}{\partial t} + \vec{u} \cdot \nabla q &= 0. & (1.5) \end{aligned}$$

In other words, these quantities q are moving within that simulation domain Ω without being necessarily modified. That being said, we will now explain the influence of each term of Eq. 1.1 to better understand the motion of each portion of fluid. The left-hand side (LHS) terms of Eq. 1.1 are respectively the acceleration, the advection, and the pressure gradient, whereas the right-hand side (RHS) terms are respectively the external forces and the viscosity. In practice, each term of Eq. 1.1 is solved separately to prevent numerically solving for a huge system, which is called *splitting*. In general, the rate of change of a physical quantity $\frac{\partial q}{\partial t}$ can be expressed as the sum of several terms. For example, any of the aforementioned pairs of terms of Eq. 1.1 can be summarized in two consecutive steps as follows:

$$\begin{aligned} \tilde{q} &= q^n + \alpha \delta t, & (1.6) \\ q^{n+1} &= \tilde{q} + \beta \delta t, & (1.7) \end{aligned}$$

where q is a quantity computed at time n and δt is the time step between two consecutive frames. Within these steps, the intermediate contribution of Eq. 1.6 is added to the contribution of Eq. 1.7 (similar to the first-order forward Euler method). Using this *splitting* approach with Eq. 1.1, each term is added to compute the resulting velocities that are used to update the position of the fluid elements. The stability and precision of this step highly depend on the integration method used. For example, a simple forward Euler integration step can be expressed as follows:

$$x_{t+1} = x_t + \triangle t \dot{x}_t, \qquad (1.8)$$

in which the integration error is $(\triangle t)^2$ per time step (i.e., $\left(\frac{\triangle t}{k}\right)^2 k$ for the whole simulation, where k equals the number of time steps). A smaller time step can reduce this error at the cost of considerably lengthening the time necessary to compute a single frame. Other explicit integration methods offer more stability and precision (e.g., Runge-Kutta and Midpoint).

Lastly, as mentioned earlier, in computer graphics we use the incompressible form of the Navier-Stokes equations. We assume that the fluid mass and density will not change much over time, especially for visual purposes. This is where the second equation (Eq. 1.2) comes into play. This equation is called the *incompressibility condition* which enforces that the velocity within the simulation domain remains divergence-free.

1.1.2. Fluid Flow Viewpoints

A broad set of numerical methods has been proposed over the last few decades to discretize fluids. These methods can be generally grouped into two main categories: Lagrangian and Eulerian. There is also a third category combining the two methods, which is generally referred to as hybrid.

Lagrangian methods treat the elements of liquid as particles. Each particle has a position $\vec{x}$ and a velocity $\vec{u}$ (bottom image of Fig. 1.1). Lagrangian methods are capable of reproducing thin structures such as liquid sheets, droplets, and splashes. Although these methods can provide efficient and intuitive ways to focus on the fluid elements within a constrained simulation domain Ω, they require a more complex data structure to encode their neighborhood connectivity. Among several methods available, Smoothed Particle Hydrodynamics (*SPH*) [12, 27] and Position-Based Fluids (*PBF*) [25] are arguably the most widely used by pure Lagrangian schemes. In the following section, we describe in more depth the *SPH* method exploited by the hybrid method proposed in our first contribution.

Smoothed Particle Hydrodynamics methods are essentially based on interpolating quantities evaluated at arbitrary locations in space. Algo. 1 shows the main steps calculated by a traditional *SPH* method at each time step $\triangle t$ of the simulation. These approximated quantities are computed with contributions from a finite number of neighbor particles. The interpolation of a quantity A_i evaluated at an arbitrary location $\vec{r_i}$ for a particle i is computed from known quantities A_j of neighboring particles j at location $\vec{r_j}$:

$$A_i(r) = \sum_j \frac{m_j}{\rho_j} A_j W(\vec{r_i} - \vec{r_j}, h), \tag{1.9}$$

where m_j is the mass of particle j , ρ_j the density at the position of particle j, and function $W(\vec{r}, h)$ the smoothing kernel evaluating the contribution between particles i and j. In

Algorithm 1: Overview of a traditional *SPH* method.

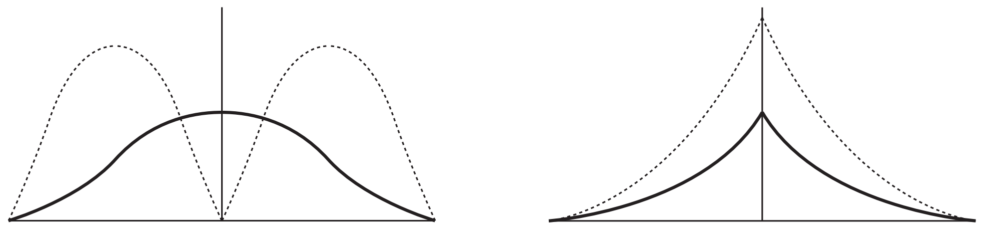
```
Input: P                                                    /* set of particles */
for $i \in \mathbf{P}$ do
    $\rho_i = \sum_j m_j W_{ij}$                             /* density at particle $i$ */
    $p_i = k\left(\left(\frac{\rho_i}{\rho_0}\right)^7 - 1\right)$          /* Tait's pressure calculation */
for $i \in \mathbf{P}$ do                                   /* sum forces at particle $i$ */
    $\vec{F}_i^{\mathrm{p}} = -\frac{m_i}{\rho_i}\nabla p_i$                    /* pressure forces */
    $\vec{F}_i^{\nu} = m_i \nu \nabla^2 \vec{u}_i$                              /* viscosity forces */
    $\vec{F}_i^{\mathrm{ext}} = m_i \vec{g}$                    /* external forces (e.g., gravity) */
    $\vec{F}_i = \vec{F}_i^{\mathrm{p}} + \vec{F}_i^{\nu} + \vec{F}_i^{\mathrm{ext}}$
for $i \in \mathbf{P}$ do
    $\vec{u}_i(t + \triangle t) = \vec{u}_i(t) + \triangle t \frac{\vec{F}_i}{m_i}$              /* velocity update */
    $\vec{x}_i(t + \triangle t) = \vec{x}_i(t) + \triangle t \vec{u}_i(t + \triangle t)$          /* position update */
$\triangle t = \lambda \frac{h}{||\vec{u}_{\max}||}$                 /* adaptive time step (Eq. 1.22) */
```

Eq. 1.9, function $W(\vec{r},h)$ is evaluated using a smoothing radius h and is expressed as follows:

$$W(\vec{r},h) = \frac{1}{h^d} f(\vec{r}), \tag{1.10}$$

where d is the simulation domain dimensions, and f is the desired smoothing kernel. The smoothing kernel is usually similar to a Gaussian function [27]. The idea is to give less importance to particles j farther from the estimated quantity at position i, and vice versa. The number of neighboring particles that are considered in the summation of Eq. 1.9 is dependent on the spacing between particles (initialized by the user), the smoothing kernel $k(\vec{r})$, and the domain dimensionality d. The spacing between particles is usually of the same order of magnitude as the smoothing radius h [27]. The chosen smoothing kernel function influences the approximation obtained by the summation. A typical smoothing kernel function is the cubic spline:

$$W_{\text{cubic}}(\vec{r},h) = \frac{1}{\pi h^3} \begin{cases} 1 - \frac{3}{2} - \left(\frac{\vec{r}}{h}\right)^2 + \frac{3}{4}\left(\frac{\vec{r}}{h}\right)^3 & 0 \leq \frac{\vec{r}}{h} < 1 \\ \frac{1}{4}(2 - \frac{\vec{r}}{h})^3 & 1 \leq \frac{\vec{r}}{h} < 2 \\ 0 & \text{otherwise.} \end{cases} \tag{1.11}$$



(a) *poly6* (b) *spiky*

Figure 1.2. Comparison between the influence of smoothing kernel functions (a) *poly6* and (b) *spiky*. The bold curve represents the function and the dashed curve its gradient.

As shown in Fig. 1.2, there are other smoothing kernel functions used in fluid simulation [30, 8]:

$$W_{poly6}(\vec{r},h) = \frac{315}{64\pi h^9}\begin{cases}(h^2-\vec{r}^{\,2})^3 & 0\leq\vec{r}\leq h\\ 0 & \text{otherwise.}\end{cases} \quad (1.12)$$

$$W_{spiky}(\vec{r},h) = \frac{45}{\pi h^6}\begin{cases}(h-\vec{r})^3 & 0\leq\vec{r}\leq h\\ 0 & \text{otherwise.}\end{cases} \quad (1.13)$$

For example, some kernels are better suited to approximate certain forces. The advantage of the *poly6* [30] smoothing kernel is that it does not require the additional computation of a square root (between particle i and its neighbors j). Nevertheless, this type of smoothing kernel may be inadequate for estimating pressure forces. In this case, particles tend to form agglomerations for high-pressure values. The closer the particles come together, the more the pressure forces dissipate (gradient tends towards zero). Using the *spiky* [8] smoothing kernel resolves the dissipating gradient issue. The spatial derivatives of a quantity A for particle i are approximated by applying operators solely to the kernel function [29]. The gradient (Eq. 1.14), divergence (Eq. 1.15), and Laplacian (Eq. 1.16) operators are expressed as follows:

$$\nabla A_i = \rho_i\sum_j m_j\left(\frac{A_i}{\rho_i^2}+\frac{A_j}{\rho_j^2}\right)\nabla W_{ij}, \quad (1.14)$$

$$\nabla\cdot\vec{A}_i = -\frac{1}{\rho_i}\sum_j m_j\vec{A}_{ij}\cdot\nabla W_{ij}, \quad (1.15)$$

$$\nabla^2 A_i = 2\sum_j\frac{m_j}{\rho_j}A_{ij}\frac{\vec{x}_{ij}\cdot\nabla W_{ij}}{\vec{x}_{ij}\cdot\vec{x}_{ij}+0.01h^2}, \quad (1.16)$$

where the index ij denotes a difference between a value evaluated at position i and a value evaluated at position j (e.g., $\vec{r_{ij}} = |\vec{r_i} - \vec{r_j}|$). In addition, the gradient of W_{ij} is expressed as $\left(\frac{\partial W_{ij}}{\partial r_{i_x}}, \frac{\partial W_{ij}}{\partial r_{i_y}}, \frac{\partial W_{ij}}{\partial r_{i_z}}\right)$. These expressions are particularly useful when computing the pressure p_i (Eq. 1.14) and viscosity ν (Eq. 1.16) forces. For example, as we need to evaluate the pressure gradient to make the velocity field divergence-free, the gradient form will be used as follows:

$$\nabla p_i = \rho_i \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right) \nabla W_{ij}, \tag{1.17}$$

where the pressure term p_i can be computed in many ways. A popular and stable form of the constitutive equation is Tait's equation [28]:

$$p_i = k\left(\left(\frac{\rho_i}{\rho_0}\right)^7 - 1\right), \tag{1.18}$$

where k is the stiffness of the fluid, ρ_0 the rest density, and ρ_i the density at the location of a particle i:

$$\rho_i = \sum_j m_j W_{ij}. \tag{1.19}$$

Enforced by its Lagrangian nature, the *SPH* method requires knowledge of the neighborhood for each particle. A neighborhood structure is used to approximate the physical properties of the simulation by a weighted summation. Naturally, at each time step $\triangle t$, this neighborhood changes dynamically following the advection of the particles. Although an adaptive structure such as a *kd-tree* seems appropriate tp efficiently keep track of neighborhoods, many prefer the use of a uniform grid. A uniform grid is subdivided into cells of the same size. This way the grid is constructed in $\mathcal{O}(n)$ and is accessed in constant time, i.e., $\mathcal{O}(1)$. Other types of hierarchical structures require $\mathcal{O}(n\log(n))$ at creation and $\mathcal{O}(\log(n))$ in access, which may explain the implementation choice of using a uniform structure. Structures of type *kd-tree* may be preferred in a context of non-uniform distributions. Finally, once the divergence-free velocity field is obtained, the position and velocity of each particle i is updated using a semi-implicit integrator of Euler (also called *symplectic Euler*):

$$\vec{u}_{t+\triangle t} = \vec{u}_t + h\dot{\vec{u}}_t, \tag{1.20}$$

$$\vec{x}_{t+\triangle t} = \vec{x}_t + h\vec{u}_{(t+\triangle t)}. \tag{1.21}$$

There are other time integrators available, but the symplectic one is mostly favored for Lagrangian approaches [35, 15]. The acceleration force $\dot{u}_t$ is calculated from the pressure, viscosity, and external forces at time t. The adaptive time step $\triangle t$ is constrained by the Courant-Friedrich-Levy (CFL) condition:

$$\triangle t = \lambda \frac{h}{||\vec{u}_{\max}||}. \tag{1.22}$$

The constant λ is initialized to 0.4 [27]. The velocity $\vec{u}_{\max}$ is expressed as the maximum velocity allowed at a time t. This time step $\triangle t$ is said to be adaptive since it can be recomputed at each simulation iteration.

In our first contribution [33], we propose an approach to couple the *SPH* method to an Eulerian grid. The main known limitations of the *SPH* method over the *FLIP* method are the stability and the longer computation time required per simulation iteration. By its stability, the *FLIP* method makes it possible to use much larger time steps, thus reducing considerably computation times per frame. Although the stability of *SPH* methods has been significantly improved in the last years (e.g., *DFSPH* [2]), computation times per frame remain much higher than with the traditional *FLIP* method. Inspired by the recently introduced narrow band *FLIP* [11], we propose a hybrid method defining the liquid interface as a band of *SPH* particles. By handling the coarse portions of the liquid volumes with a uniform Eulerian grid, we are able to offer significant speed-up factors over the state of the art while preserving fine details making the *SPH* model realistic.

Eulerian methods, as opposed to Lagrangian methods, track fluid quantities (e.g., velocity $\vec{v}$, density ρ, etc.) at fixed positions in space (top image of Fig. 1.1). These positions are usually defined by cells of a uniform grid structure. An advantage of using an Eulerian viewpoint to discretize fluids is that spatial derivatives (e.g., pressure gradients) are much easier to approximate numerically using uniform and regular data structures, which are also more suitable for GPU implementation. However, thin liquid structures remain difficult to capture and model correctly for details below the resolution of the underlying grid cell. In computer graphics, Stam's work [37] remains a reference when it comes to Eulerian methods for fluids. The implementation details of a grid-based liquid simulation is similar to the Algo. 2 presented for the *FLIP* method later in Section 1.2.1. However, as opposed to *FLIP*, which advects particles, a *level set* ϕ serves as the free surface to represent liquids in purely Eulerian methods.

1.2. Hybrid Methods for Efficient and Detailed Liquids

As previously exposed, Lagrangian and Eulerian methods have their advantages and disadvantages. This fact has inspired researchers to combine both viewpoints (Fig. 1.3) to overcome these known limitations while making the most of their strengths. In the following sections, we provide a brief overview of the different ways to couple these schemes in order to improve previous work. They are also related to our first contributions presented in Chapter 2.

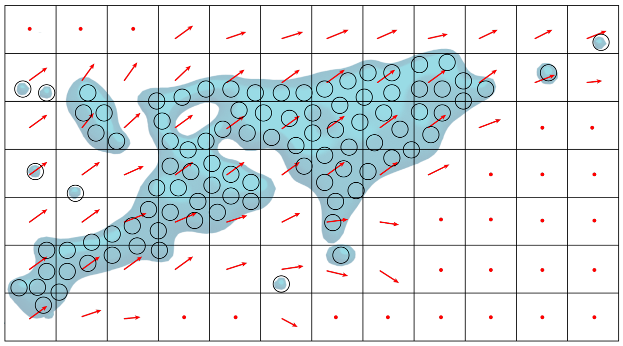


Figure 1.3. Visualization of a hybrid method of the same animation presented in Fig. 1.1.

1.2.1. Combining Methods

By combining methods, it is possible to take advantage of the various structures that make up the simulation model. Using both particles and a data structure such as a uniform grid, fluid elements can be defined using Lagrangian and Eulerian quantities. In general, hybrid methods are intended either to speed up numerical solvers or to reproduce the details that would otherwise be dissipated. A well-known and most commonly used hybrid method in the visual effects community is the Fluid Implicit Particle (FLIP) [45]. Several hybrid methods combining particles with different types of Eulerian data structures have been introduced over the last two decades. Often used as computational structures, these types of Eulerian data structures include: 3D MAC grids [17], 2D height fields [18], meshes [20, 4], and so on. One of the existing challenges on these hybrid methods is to couple both schemes, enforcing stability while preserving small-scale details during simulation as we do in our hybrid Eulerian-*DFSPH* approach (Chapter 2). In the following section, we describe in more depth the FLIP method as exploited by our second [32] and third contribution [34].

Fluid Implicit Particle methods are the most commonly used in the visual effects industry. The method was introduced to computer graphics in order to simulate sand [45]. The *FLIP* method takes advantage of the Lagrangian and Eulerian viewpoints. This method allows a Lagrangian advection on a set of particles while offering to solve the interaction with boundaries as well as the incompressibility of the fluid using a grid. The standard steps of a *FLIP* simulation are presented in Algo. 2. The implementation proposed in this document (and in our work) uses a level set to initialize the volume of liquid.

Algorithm 2: Overview of a traditional *FLIP* method.

```
        liquid     /* initialize the liquid volume with an input mesh */
Input:  obstacles  /* initialize static boundaries with input meshes */
        resolution       /* discretization of the velocity grid $\mathbf{u}_{\text{MAC}}$ */
$\mathbf{u}_{\text{MAC}}$ = initializeGridMAC(resolution);
$\mathbf{P}$ = sampleGridWithParticles($\mathbf{u}_{\text{MAC}}$);
$\phi_l$ = computeLevelSet(liquid)
for $o_i \in$ obstacles do
    $\phi_o \cup$ computeLevelSet($o_i$)
flags = updateFlagCells($\phi_l$,$\phi_o$)
for $f_i \in$ frames do
    advectInGrid($\mathbf{P}$, $\mathbf{u}_{\text{MAC}}$, $\triangle t$)
    weights = mapParticlesToGrid($\mathbf{P}$, $\mathbf{u}_{\text{MAC}}$, flags)
    extrapolateGridWeight($\mathbf{u}_{\text{MAC}}$, weights)
    updateFlagCells($\mathbf{P}$, flags)           /* update fluid cells */ applyGravity($\mathbf{u}_{\text{MAC}}$)
    updateBoundaryConditions($\mathbf{u}_{\text{MAC}}$, flags)
    solvePressure($\mathbf{u}_{\text{MAC}}$, flags)
    applyVelocityUpdate($\mathbf{P}$, $\mathbf{u}_{\text{MAC}}$)
```

A grid of labels (*flag* grid) is used to identify the nature of the elements contained in each cell (i.e., fluid, air, and solid). The information about the simulation is stored in a *MAC* grid (*Marker-and-cell*). This type of grid allows preserving velocities on the faces of cells and pressure forces at the center of cells (Fig. 1.4). Using a uniform grid whose values are all calculated at the center of cells is not ideal [3], especially at borders. For example, the divergence calculation does not take into account the current cell. The divergence at the

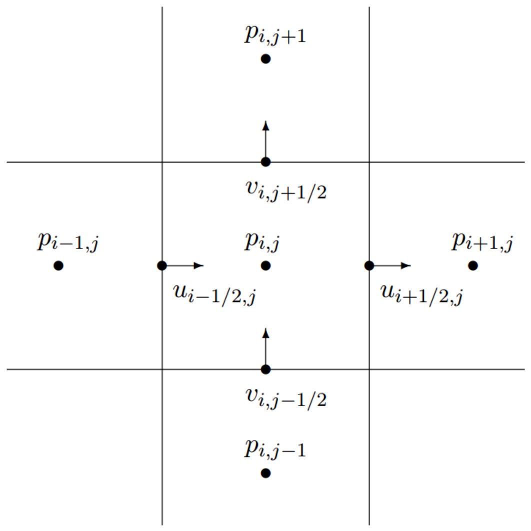


Figure 1.4. An example of the *MAC* grid in 2D storing pressures p and velocity components u and v [3].

center of cells (of dimension $\triangle x$) is defined by the expression:

$$\nabla \cdot \vec{u}_{i,j} = \frac{\vec{u}_{i+1,j} - \vec{u}_{i-1,j} + \vec{u}_{i,j+1} - \vec{u}_{i,j-1}}{2\triangle x}. \tag{1.23}$$

This calculation method can generate large differences between divergence values in the grid. The *MAC* grid allows one to work around this problem by reformulating Eq.1.23 into:

$$\nabla \cdot \vec{u}_{i,j} = \frac{\vec{u}_{i+\frac{1}{2},j} - \vec{u}_{i-\frac{1}{2},j} + \vec{u}_{i,j+\frac{1}{2}} - \vec{u}_{i,j-\frac{1}{2}}}{\triangle x}. \tag{1.24}$$

Another important part is the transfer of velocities to the grid (and vice versa). The transfer uses a weighted interpolation on the particles inside the current cell. Cells that contain particles are flagged to speed up transfer and force calculations. The interpolation is computed relative to each corner vertex of a cell. Each value weighted by the distances between the corner vertices is stored at the faces of the *MAC* grid. In the *FLIP* method, all forces are stored in the grid. The advection step is the only one performed directly on the particles. The pressure solve is performed directly on the grid $\mathbf{u}_{\text{MAC}}$. This step is called the projection step and is required to satisfy the condition of incompressibility (Eq. 1.2). In theory, we need to determine the Lagrange multipliers whose gradient subtractions give a velocity field without divergence. The equation is in the form of a Poisson equation. One way to formulate it is by the Helmholtz-Hodge decomposition. This formulation highlights the fact

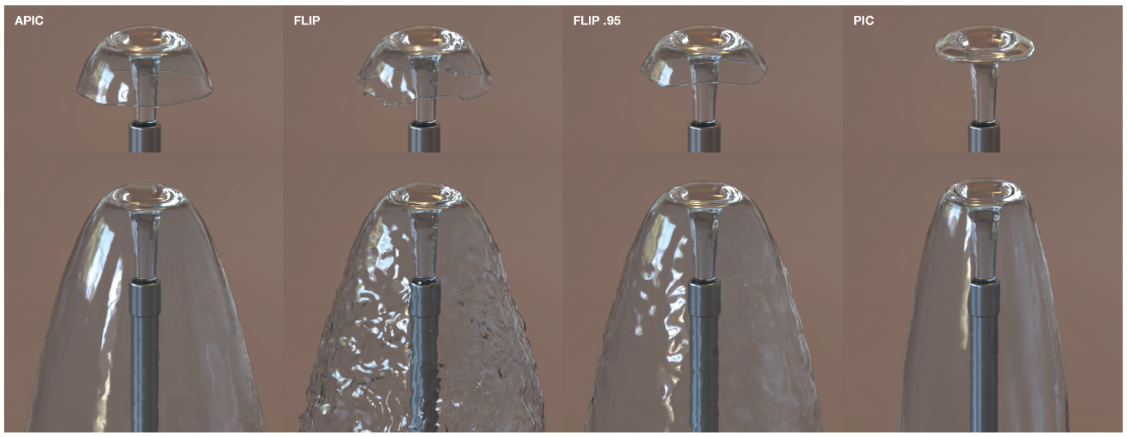


Figure 1.5. Comparing *Affine-PIC* (*APIC*), *FLIP*, *PIC-FLIP*, and *PIC*. The *APIC* method considerably reduces the noise generated by *FLIP* while preserving the nature of the fluid [17].

that most vector fields can be represented by a sum of vector fields without divergence and an irrotational field. It is therefore sufficient to subtract the pressure gradient from the velocities to ensure the conservation of volume. The system to be solved is the following Poisson equation:

$$\nabla^2 p = \nabla \cdot \vec{u}. \tag{1.25}$$

There are several numerical methods for solving linear systems in this form. When implementing this method, we favored the conjugate gradient as it offers fairly fast convergence times. Finally, the pressures obtained allow the velocities to be corrected in order to obtain a vector field without divergence. These velocities can now be transferred back to the particles. The interpolation method used is the same as for the transfer to the grid. However, the method of updating the interpolation velocities for a non-viscous liquid is usually a combination of *PIC-FLIP* (Eq. 1.26 and Eq. 1.27, respectively).

$$\vec{u}_i^{\text{PIC}} = interp(\vec{u}_{\text{MAC}}^{t+\triangle t}, \vec{x}_{p_i}), \tag{1.26}$$

$$\vec{u}_i^{\text{FLIP}} = \vec{u}_{\text{MAC}}^{t+\triangle t} + interp(\triangle\vec{u}_{\text{MAC}}, \vec{x}_{p_i}), \tag{1.27}$$

where $\vec{x}_i$ and $\vec{u}_i$ are respectively the position and the velocity of the particle i. The *FLIP* update method is more suitable for a liquid, but less stable than the *PIC* method (numerical dissipation). We therefore give more importance to $\vec{u}^{\text{FLIP}}$ ($\alpha = 0.97$) than to $\vec{u}^{\text{PIC}}$ $(1-\alpha)$.

Using these weights, the velocity update can be expressed for a stable and non-viscous liquid:

$$\vec{u}_i = (1-\alpha)\vec{u}_i^{\text{PIC}} + \alpha\vec{u}_i^{\text{FLIP}}. \tag{1.28}$$

More recently, an improvement of the *FLIP* method, so-called the *Affine-PIC* method, has been proposed to prevent using a weighted velocity update [17] as shown in Eq. 1.28. The main contributions of the *Affine-PIC* method are improving stability (caused by the *FLIP* velocity update) while reducing energy dissipation (compared to the *PIC* velocity update). Conserving angular momentum in the grid preserves energy at both small and large scales. A combination of the two velocity update models results in a much smoother and more stable fluid (Fig. 1.5). Later, the *FLIP* method was extended to focus the computation efforts on particles at the surface of the liquid volume and around static and dynamic boundaries [11]. Their band method resamples particles only within a narrow band near the surface. They also introduce a coupling scheme between particles and the grid to reduce energy fluctuations.

1.2.2. Adaptive Methods

As previously mentioned in Section 1.1.2, grid-based methods (i.e., Eulerian) have difficulties in resolving fluid details below the resolution of the underlying grid cell. In general, two ways have been proposed to overcome this issue: using an adaptive data structure [24] or using multiple scales of particles [36]. In both ways, the key is to detect when to increase or decrease adaptive elements while preserving visible details. Grid-based methods were the first to introduce adaptivity to alleviate the resolution bounding issues. When using adaptive data structures such as an octree, the fluid quantities (e.g., such as pressure gradients) are computed using different grid cell sizes. In these cases, one way is to compute the weighted average using a linear interpolation between cells. On the other hand, dealing with many scales of particles can also be challenging since each one of them may convey different contributions to one another. As proposed by Solenthaler and Gross [36], using a pre-determined number of scales for particles can simplify the simulation model while offering interesting features. At the other end of the spectrum, allowing the use of continuous particle scales can provide improved accuracy and stability, as recently introduced by Winchenbach et al. [41, 42].

1.3. Enriching Liquid Animations

Although hybrid and adaptive methods can alleviate limitations related to coarse discretizations, other complementary (i.e., in the simulation loop) and post-processing methods are often required to simulate small-scale phenomena such as splashes, whitewater, foam, and spray in addition to the input simulation. In general, these additional details are computed either by generating high-frequency features or by extracting fluid elements dissipated by a coarse discretization.

1.3.1. Generating High-Frequency Details

Generating high-frequency details using additional simulation structures, such as surface points, can enhance the level of detail even when using a coarse simulation resolution (i.e., low-resolution velocity grid and/or a small number of particles). Using an existing liquid animation as input to these complementary methods, fine details are seamlessly integrated, improving the flow behavior using previously computed physical characteristics. Among several others, the most commonly used input characteristics for such methods are velocity and vorticity fields. The generated details are often represented as an additional set of particles governed by constraints [10], combined forces [14], and wave simulations [26, 44]. Most of these methods are so-called *physically plausible* (as opposed to *physically accurate*) because they infer realistic details while not necessarily preserving divergence-free flows.

1.3.2. Extracting Dissipated Details

In computational fluid dynamics (CFD), numerical dissipation is referred to as the error introduced throughout iterations over a discretized simulation domain Ω, and cumulated at each time step (i.e., determined by the time integration scheme). Numerical dissipation in fluid simulations is mostly noticeable in small structures such as vortices.

Extracting dissipated features to enhance fluid details has also been an active research topic in the last decade or so. Often referred to as *vorticity confinement* methods, these techniques introduce rotational forces, allowing vortex structures to appear within the coarse flow. As introduced for visual effects by Lentine et al. [22], vorticity confinement methods allow for preserving both energy and momentum present in a coarse simulation. Such vortex structures can be expressed using multiple energy scales (sub-grid fluid values) [16] or by amplifying existing vorticity [25]. In general, vorticity confinement methods provide ways for preserving small-scale vortices but are unable to reproduce surface turbulence as opposed to the high-frequency-based methods aforementioned in Section 1.3.1.

1.4. Machine Learning Methods for Liquid Simulations

As part of the extended family of simulated natural phenomena, fluid simulations have become in recent years a clear target for machine learning. Even today, fluid simulation models remain computationally expensive and challenging to control within an artistic process for visual effects. In the last decade, many research work have been proposed to improve their underlying numerical solvers, compute complex features, and infer additional or dissipated visual details by leveraging machine learning algorithms. In the following, we briefly overview the main challenges when using machine learning to improve physically-based simulation methods.

1.4.1. Neural Networks and Physics-Based Simulations

With available ways to facilitate the generation of synthetic data, deep learning algorithms have become an effective way to learn and encode complex representations through large datasets using artificial neural networks. Deep learning algorithms allow us to map input samples X to the approximated target outputs Y through generating a learned parametric function f_θ as follows:

$$f_\theta(X) \approx Y, \tag{1.29}$$

where θ contains the learned function parameters commonly called *weights*. While predicted outputs Y (i.e., the last layer of the deep network) are generated using input samples X (i.e., the first layer of the deep network), several inner layers of neurons are used to learn these weights θ. To summarize, the inner layers can expressed as:

$$y_i = \sigma(W y_{i-1} + b), \tag{1.30}$$

where y_{i-1} is the output from the previous layer $i-1$, y_i is the output of the current layer i, and θ are the trainable weights along with biases b. The nonlinearity is introduced by the activation function σ (e.g., $\sigma_{\text{ReLU}}(x) = \max(0,z)$) to approximate the function f during the forward pass. The weights θ and biases b are refined using a loss function L through the backpropagation pass (short for backward propagation of errors) performed with an optimization method such as the gradient descent. In general, the loss function L is computed by measuring the differences between the predicted outputs and ground truth samples. The weights θ are adjusted by the gradient of the loss function ∇L (i.e., computed by the optimization method) using the chain rule. As illustrated in Fig. 1.6, the step of the optimization method is determined with a learning rate α_t initialized and updated throughout the iterations, typically using an adaptive method such as the ADAM optimizer [19]. At the end of an epoch of training, a local minimum is approximated from a cumulative loss term. One epoch means that the optimizer has passed through every training sample once. In physics-based simulation, machine learning methods are generally used to learn and im-

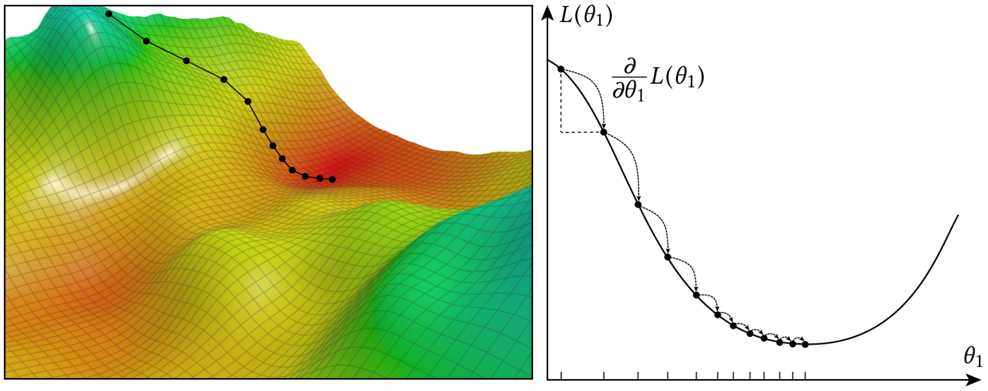


Figure 1.6. Example of the gradient descent method minimizing a single weight θ_1.

prove models in three different ways [40]: *data-driven*, *loss-driven*, and *simulation-driven*. Using one of these approaches may depend on the available data and the desired objectives. *Data-driven* approaches are used when the predicted output is generated from existing or captured simulated data. This type of approach is categorized as a forward problem as it predicts outputs directly, whereas approaches categorized as a reverse problem are predicting parameters to approximate simulated output. The second type *loss-driven* is partially integrated in the simulation model. With this type of approach, the learning process is performed, throughout the loss function, using differentiable PDE-based formulations from the simulation (e.g., from the numerical solver). In contrast, the last type, *simulation-driven*, interleaves the learning process with the simulation model. As an example, a neural network architecture (or stacked networks) is used to approximate a velocity field as a preconditioner input to the advection step of the simulation model. Given the amount of available fluid simulation data [9], *data-driven* and *loss-driven* methods for fluid simulations have become more attractive for the visual effects community [6, 39]. Although these types of methods offer convincing results, tightly integrating neural networks into the simulation model can provide more flexibility and control [21, 38]. As exposed by the previous work, each simulation viewpoint can benefit from *simulation-driven* solutions. In addition to facilitating the numerical approximation of spatial derivatives, Eulerian approaches offer intuitive capabilities when used with 2D convolution operations (such as in CNNs). On the other hand, Lagrangian approaches partially benefit from the recent advances in deep learning for point clouds.

1.4.2. Encoding High-Resolution Features for Particle-Based Liquids

In contrast to *data-driven* solutions, encoding high-resolution features is used to generate additional or dissipated details to enhance the visual appearance of existing liquid animations. This task is usually categorized into two types of learning approaches: *detail synthesis* [6] and *super-resolution* [43, 1]. With the *detail synthesis* learning approaches, a high-resolution dataset is used to learn a mapping to the corresponding low-resolution samples. In comparison, *super-resolution* approaches upscale a coarse input to infer the sub-grid details at high resolution. Xie et al. [43] propose a method using GAN to produce volumetric high-resolution details using spatial and temporal discriminators. Recently, a dictionary learning method has been proposed [1] to upsample coarse smoke flows by generating sparse representations of local velocity patches to encode high-resolution temporal features. Our contributions presented in Chapter 4 fits in the *detail synthesis* category.

As we focus on convolutional neural networks in our work in Chapter 4, understanding the fundamental differences of the discrete convolution operations on different types of data is crucial to appreciate our related contributions. Using 2D convolutions on images is rather intuitive. As shown in Fig. 1.7(a), a 3×3 filter kernel w is used on samples x. In this case, the convolution output y is performed by applying the kernel matrix w to extract various features from an input image. For CNNs, the kernel values of w are learned to extract latent features. Stacking these convolution operations per layer allows us to produce a hierarchical decomposition of this input. It is worth mentioning that images (rendered or captured) may introduce undesired features during the training process caused by capture noise or rendering artifacts, as slightly noticeable with the aliasing effect at the silhouette of the sphere in the

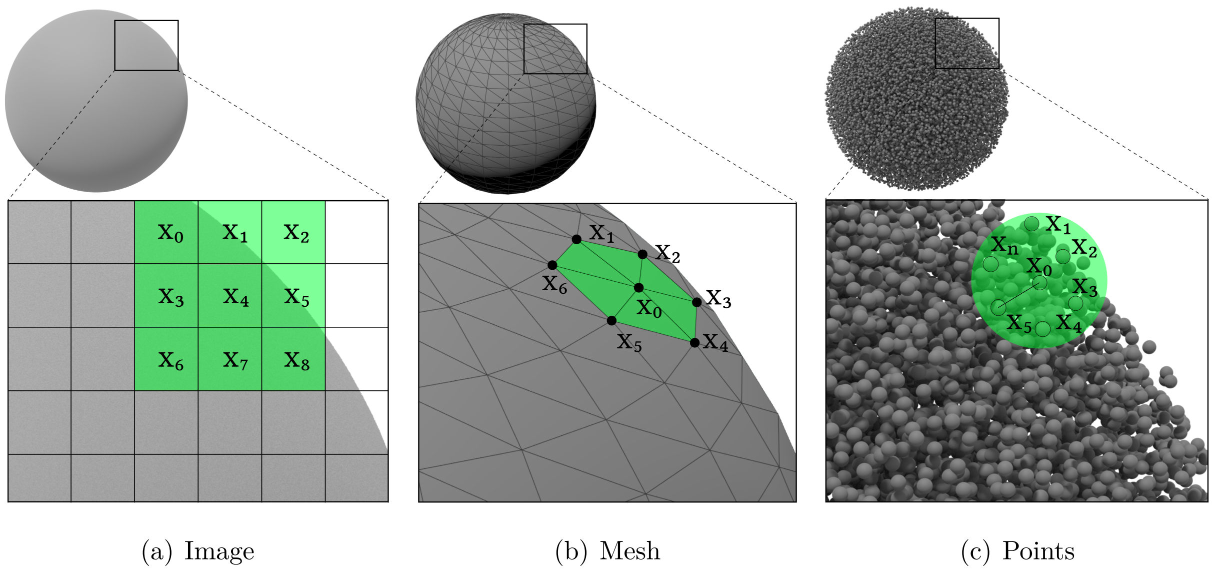


Figure 1.7. Comparing discrete convolution operations on different data structures.

enlarged view of Fig. 1.7(a). Using convolution operations on images is straightforward since they are performed on a regular grid of pixels for which the neighborhood is structured (i.e., always with the same number of neighbors and using the same relative locations). However, things become more challenging when dealing with 3D irregular structures such as meshes and point clouds. These structures are not defined on a regular grid, meaning that for each element (i.e., vertex or point), the number of neighbors and their locations might change (Figs. 1.7(b) and 1.7(c)). We refer to these structures as irregular. Different ways have been introduced to perform convolution operations on irregular structures such as meshes. In their work, Hanocka et al. [13] propose to express convolution operations on triangular meshes using intrinsic geodesic properties. Convolutions are applied to the edges allowing a fixed number of neighboring faces and associated edges to be processed. Using the mesh connectivity for such a task is important to query the neighborhood for each edge. Although their work has considerably improved our knowledge on processing irregular structures for deep learning, point clouds carry an additional layer of complexity in order to be used with learning algorithms. As illustrated in the enlarged view of Fig. 1.7(c), the number of sampled neighbors surrounding each point is highly variable (e.g., neighborhood deep under the surface versus at the surface). In contrast with meshes, point clouds (or particle systems) contain under-the-surface elements. These invisible elements must be considered during the convolution operations to fully encapsulate the contributions. Several methods have been proposed to take into account this type of data structure in 3D for deep learning algorithms [31, 23].

In comparison to Chu et al. [6], our approach is performed directly on the fluid elements in 3D, which slightly complicates the downsampling and upsampling operations. As our third contribution aims to apply displacements directly onto the particles, our network needs to handle convolution similarly to point clouds (Fig. 7(c)), hence handling neighborhoods with beneath-the-surface elements as contributions. However, since applying displacements on elements beneath the surface might introduce undesirable noise during inference, most of our training samples were generated using the narrow band *FLIP* method exploiting only a band of visible particles at the interface between the liquid and air. Nevertheless, as we focus the learning towards particle-based liquids and not point clouds, we introduced the known concept of weighted neighborhood contributions directly in the upsampling and downsampling operations to preserve the inherent properties of the liquid through embedding. We refer the reader to Chapter 4 for further details on our neural representation for particle-based liquids.

References

[1] Kai Bai, Wei Li, Mathieu Desbrun, and Xiaopei Liu. Dynamic upsampling of smoke through dictionary-based learning. *ACM Trans. on Graphics (TOG)*, 40(1):1–19, 2020.

[2] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 147–155. ACM, 2015.

[3] Robert Bridson. *Fluid Simulation for Computer Graphics.* CRC Press, 2015.

[4] Nuttapong Chentanez, Bryan E Feldman, François Labelle, James F O'Brien, and Jonathan R Shewchuk. Liquid simulation on lattice-based tetrahedral meshes. In *Proc. Symposium on Computer Animation*, pages 219–228. ACM SIGGRAPH/Eurographics, 2007.

[5] Nuttapong Chentanez, Matthias Müller, and Tae-Yong Kim. Coupling 3D Eulerian, heightfield and particle methods for interactive simulation of large scale liquid phenomena. *IEEE Trans. on Visualization and Computer Graphics*, 21(10):1116–1128, 2015.

[6] Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Trans. on Graphics (TOG)*, 36(4), 2017.

[7] Mathieu Desbrun. *Space-time adaptive simulation of highly deformable substances.* PhD thesis, INRIA, 1999.

[8] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation'96*, pages 61–76. Springer, 1996.

[9] Marie-Lena Eckert, Kiwon Um, and Nils Thuerey. Scalarflow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Trans. on Graphics (TOG)*, 38(6), 2019.

[10] Gang Feng and Shiguang Liu. Detail-preserving SPH fluid control with deformation constraints. *Computer Animation and Virtual Worlds*, 29(1):e1781, 2018.

[11] Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. Narrow band FLIP for liquid simulations. *Computer Graphics Forum*, 35(2):225–232, 2016.

[12] Robert A. Gingold and Joseph J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181(3):375–389, 1977.

[13] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Trans. on Graphics (TOG)*, 38(4):1–12, 2019.

[14] Markus Ihmsen, Nadir Akinci, Gizem Akinci, and Matthias Teschner. Unified spray, foam and air bubbles for particle-based fluids. *The Visual Computer*, 28(6):669–677, 2012.

[15] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible SPH. *IEEE Trans. on Visualization and Computer Graphics*, 20(3):426–435, 2014.

[16] Taekwon Jang, Heeyoung Kim, Jinhyuk Bae, Jaewoo Seo, and Junyong Noh. Multilevel vorticity confinement for water turbulence simulation. *The Visual Computer*, 26(6):873–881, 2010.

[17] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Trans. on Graphics (TOG)*, 34(4):1–10, 2015.

[18] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Proc. ACM on Computer Graphics and Interactive Techniques*, pages 49–57, 1990.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[20] Bryan M Klingner, Bryan E Feldman, Nuttapong Chentanez, and James F O'Brien. Fluid animation with dynamic meshes. *ACM Trans. on Graphics (TOG)*, 25(3):820–825, 2006.

[21] L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Trans. on Graphics (TOG)*, 34(6), 2015.

[22] Michael Lentine, Mridul Aanjaneya, and Ronald Fedkiw. Mass and momentum conservation for fluid simulation. In *Proc. Symposium on Computer Animation*, pages 91–100. ACM SIGGRAPH/Eurographics, 2011.

[23] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3D point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 529–537, 2019.

[24] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. on Graphics (TOG)*, 23(3):457–462, 2004.

[25] Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. on Graphics (TOG)*, 32(4):1–12, 2013.

[26] Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim, and Derek Nowrouzezahrai. Surface turbulence for particle-based liquid simulations. *ACM Trans. on Graphics (TOG)*, 34(6):1–10, 2015.

[27] Joe J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574, 1992.

[28] Joe J Monaghan. Simulating free surface flows with SPH. *Journal of Computational Physics*, 110(2):399–406, 1994.

[29] Joseph P Morris, Patrick J Fox, and Yi Zhu. Modeling low reynolds number incompressible flows using SPH. *Journal of Computational Physics*, 136(1):214–226, 1997.

[30] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proc. Symposium on Computer Animation*, pages 154–159. ACM SIGGRAPH/Eurographics, 2003.

[31] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[32] Bruno Roy, Eric Paquette, and Pierre Poulin. Particle upsampling as a flexible post-processing approach to increase details in animations of splashing liquids. *Computers & Graphics*, 88:57–69, 2020.

[33] Bruno Roy and Pierre Poulin. A hybrid Eulerian-DFSPH scheme for efficient surface band liquid simulation. *Computers & Graphics*, 77:194–204, 2018.

[34] Bruno Roy, Pierre Poulin, and Eric Paquette. Neural upflow: A scene flow learning approach to increase the apparent resolution of particle-based liquids. In *Proc. Symposium on Computer Animation*. ACM SIGGRAPH/Eurographics, 2021.

[35] Hagit Schechter and Robert Bridson. Ghost SPH for animating water. *ACM Trans. on Graphics*, 31(4):Art. 61, 2012.

[36] Barbara Solenthaler and Markus Gross. Two-scale particle simulation. *ACM Trans. on Graphics (TOG)*, 30(4):1–8, 2011.

[37] Jos Stam. Stable fluids. In *SIGGRAPH Conference Proceedings*, pages 121–128, 1999.

[38] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating Eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pages 3424–3433. PMLR, 2017.

[39] Maximilian Werhahn, You Xie, Mengyu Chu, and Nils Thuerey. A multi-pass GAN for fluid flow super-resolution. *Proc. ACM on Computer Graphics and Interactive Techniques*, 2(2), 2019.

[40] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating physics-based modeling with machine learning: A survey. *arXiv preprint arXiv:2003.04919*, 2020.

[41] Rene Winchenbach, Hendrik Hochstetter, and Andreas Kolb. Infinite continuous adaptivity for incompressible SPH. *ACM Trans. on Graphics (TOG)*, 36(4):1–10, 2017.

[42] Rene Winchenbach and Andreas Kolb. Optimized refinement for spatially adaptive SPH. *ACM Trans. on Graphics (TOG)*, 40(1):1–15, 2021.

[43] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Trans. on Graphics (TOG)*, 37(4), 2018.

[44] Sheng Yang, Xiaowei He, Huamin Wang, Sheng Li, Guoping Wang, Enhua Wu, and Kun Zhou. Enriching SPH simulation by approximate capillary waves. In *Proc. Symposium on Computer Animation*, pages 29–36. ACM SIGGRAPH/Eurographics, 2016.

[45] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. on Graphics (TOG)*, 24(3):965–972, 2005.

Chapter 2

A Hybrid Eulerian-DFSPH Scheme for Efficient Surface Band Liquid Simulation

This first paper introduces a hybrid approach for efficient and realistic particle-based liquid simulations. Our approach combines a low-resolution Eulerian simulation with an SPH particle-based simulation to model the liquid behavior below and at the surface, respectively. By approximating the behavior of the liquid below the surface with a coarser representation, our approach offers a significant reduction in computation times upon previous work. In addition, our SPH particle band formulation allows us to focus our attention on visible regions of the surface and therefore improves the level of detail in those volumes of liquid.

Publication

This paper [82] was published in Computers & Graphics, Volume 77, in December 2018. It has been reformatted for this thesis. Roy was involved in the formulation and numerical analysis of the hybrid approach. He was also responsible for the design, implementation, and evaluation of the approach. Poulin supervised each stage of the research project. The first draft of the published paper was written by Roy and improved by Poulin prior and during the submission process.

A Hybrid Eulerian-DFSPH Scheme for Efficient Surface Band Liquid Simulation

BRUNO ROY, *Université de Montréal*
PIERRE POULIN, *Université de Montréal*

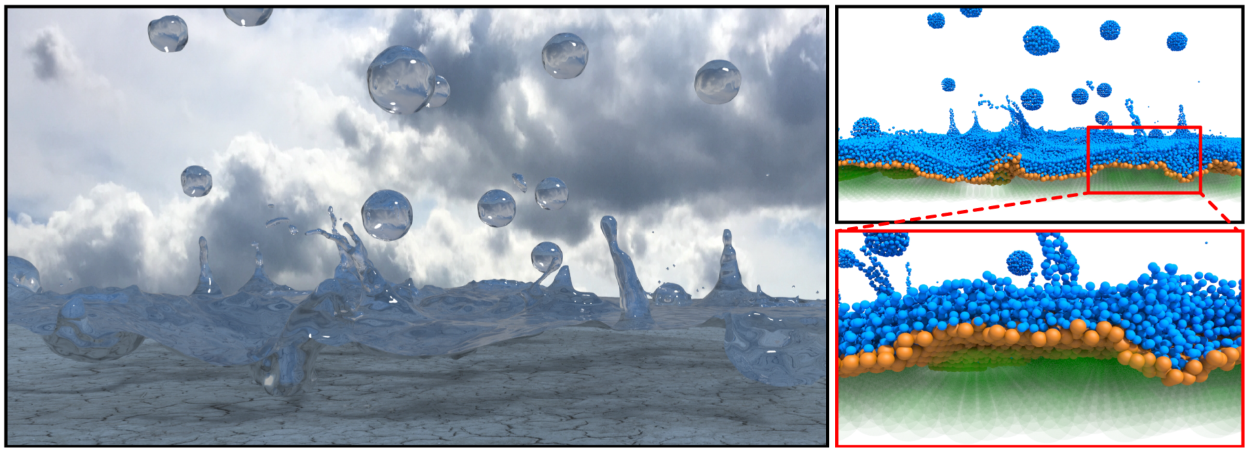


Figure 2.1. A high-quality rendered close-up view (left) of a larger scene of light rain simulated with 10M SPH particles (2M band particles and 300k fictitious particles). The scene is clipped at the front plane to show the water surface and the ground underneath. The two types of particles (band SPH in blue and fictitious in orange) for this frame are shown at the top right. The remaining portion of the liquid is handled by our Eulerian solver (grid in green). The image at the bottom right shows a sensitive region where the density contribution is computed with SPH and fictitious particles (with slightly larger masses).

Abstract

The discretization of fluids can significantly affect computation times of traditional SPH methods. Even if state-of-the-art SPH methods such as divergence-free SPH (DFSPH) produce excellent small-scale details for complex scenarios, simulating a large volume of liquid with these particle-based approaches is still highly expensive considering that only a fraction of the particles will contribute to the visible outcome. This paper introduces a hybrid Eulerian-DFSPH method to reduce the dependency of the number of particles by constraining kernel-based calculations to a narrow band of particles at the liquid surface. A coarse Eulerian grid handles volume conservation and leads to a fast convergence of the pressure forces through a hybrid method. We ensure the stability of the liquid by seeding larger

particles (which we call fictitious) in the grid below the band and by advecting them along the SPH particles. These fictitious particles require a two-scale DFSPH model with distinct masses inside and below the band. The significantly heavier fictitious particles are used to correct the density and reduce its fluctuation inside the liquid. Our approach effectively preserves small-scale surface details over large bodies of liquid, but at a fraction of the computation cost required by an equivalent reference high-resolution SPH simulation.
Keywords: Physics-based Animation, Hybrid Method, Surface Band, Particle-based Liquid

2.1. Introduction

Visual effects (VFX) in feature films are constantly raising the bar for physically based liquid simulations in terms of level of detail, computation times, and controllability. With most types of liquid simulations, the level of detail is mainly dependent on the finite resolution of the discretization for the associated structures. In the VFX industry, implicit and explicit particle-based methods are the most widely used for liquid simulations. Although Fluid-Implicit-Particle (FLIP) methods for liquid simulations are usually preferred for large-scale scenarios, SPH methods are still widely used for their realistic behavior, including the generation of natural small-scale details (as shown in Figure 2.1). SPH simulations also allow for effective modeling of surface tension and its related free-surface cohesive forces.

Unfortunately, state-of-the-art SPH methods are still expensive in term of computation compared to FLIP methods. Each particle carries a heavy cost for its contributions with neighboring particles. However, combining Lagrangian and Eulerian methods has proven to be an efficient and tailored solution to consider different scales of detail for complex scenarios in fluid simulations. Purely Lagrangian methods are well suited to handle small-scale details close to geometrically complex boundaries and close to the free surface (interface between air and liquid). In all other regions (e.g., below the free surface), an Eulerian perspective proves an appropriate alternative. In order to meet industry needs, we propose a method coupling a particle surface band with a grid-based liquid simulation. Our approach uses SPH particles within a surface band defined along the level set ϕ at the interface (i.e., where $\phi = 0$). The thickness of our surface band is dependent on the underlying support radius used by the SPH particles of the narrow band (and a user-defined distance from $\phi = 0$). Particles with larger masses are seeded close to the transition between the particle band and the Eulerian grid. Those particles are used to couple the solvers by correcting the density around this transition. We call these particles *fictitious* because their contribution is exclusively on a density-level.

The work of Chentanez et al. [57] on a hybrid grid-SPH method is the most similar to ours in spirit. They achieve to compute their model in real time but at the cost of some inherent physical differences caused by the use of Shallow Water Equations (SWE) on a 3D grid and a coupling that is highly sensitive to an adequate parameterization. In comparison, our Eulerian-based approach with fictitious particles allows us to retain the realism and performance properties provided by a state-of-the-art SPH method. The only user-defined parameter to tune is the number of fictitious particles seeded per cell along the particle band. Furthermore, as for narrow band FLIP [63], our approach will be slightly slower than a traditional SPH method if it is used for scenarios consisting of exclusively thin sheets of liquid. A minimum band thickness is required in order to exploit the benefits of the hybrid structure. However, our hybrid method allows preserving the Lagrangian nature of a band-only SPH model without suffering from the noise and instability of FLIP. With our hybrid model, we want to avoid grid artifacts such as undesired interactions between separate liquid bodies prior to collisions between them. Our method allows a drastic reduction of the number of particles compared to equivalent high-resolution liquids. In summary, the contributions of our approach consist of:

- We extend a hybrid model to focus on a band of SPH particles at the interface defined by a level set method;
- We use fictitious particles within our grid to correct for neighborhood deficiency and to couple the Lagrangian and the Eulerian solvers;
- We exploit a two-scale model to correct the densities through the liquid inside our simulation domain.

The results achieved over various scenarios show only small differences from the ground truth defined by reference simulations. Moreover, we are able to preserve the small-scale realistic behavior of an SPH method at much lower cost than a full SPH simulation.

2.2. Related Work

Animating large volumes of liquid is computationally expensive and remains a challenging problem, especially within an SPH method. Such a density-based model is highly related to neighborhood contributions. Several approaches have been introduced to reduce this influence on the computation of forces by combining a Lagrangian method with a grid-based (Eulerian) representation [77, 68, 75, 91, 81, 85, 57].

However, with most of these hybrid methods, considerable efforts are focused on regions where there are no significant visual contributions (i.e., beneath the surface). With our hybrid method, we significantly reduce the dependency of the SPH neighborhood influence to focus computation efforts almost exclusively around the surface of the liquid, and this, without introducing grid artifacts. Over the last decade, various approaches have been presented to combine fluid simulation models and exploit their respective strengths. Combining various models may be especially beneficial when there is a need to capture several distinct scales of behavior. As an introduction to this field of study, we will briefly highlight a few approaches proposed to effectively represent these various scales of detail in fluid simulations.

2.2.1. Large-scale Scenes

The FLIP method [55] is the most widely used hybrid model in fluid simulations for VFX; it was introduced in computer graphics by Zhu and Bridson [92]. Velocities are stored on particles, and pressure projection is performed on a grid. The resulting divergence-free velocity field is used to interpolate those values back to particles. The FLIP method is widely used to simulate large-scale scenarios (e.g., large bodies of liquid breaking over complex geometries). Other researchers extend the FLIP method by adaptively resampling particles near the surface to improve the level of detail in these regions [48] or by using a spatially adaptive mesh structure to capture and model the behavior of particles at different scales [49]. Um et al. [88] improve the distribution of particles around the fluid interface by applying a correction on positions in high-frequency regions within a sub-grid scale. Jiang et al. [72] introduce Affine-PIC, which proposes a way to resolve the limitations of the FLIP methods by preserving the stability of the Particle-in-cell (PIC) velocity update without dissipation. Finally, Aanjaneya et al. [46] propose a method to take advantage of a power diagram with an adaptive octree structure to preserve high-resolution details near the liquid surface.

2.2.2. Small-scale Behavior and Fine Details

SPH methods have often been preferred for their realistic small-scale behavior. However, their scheme, based on neighborhood contributions, makes them very expensive in term of computation time. The method was originally proposed [66] a while before being introduced to the computer graphics community by Desbrun and Gascuel [62], and later improved by Müller et al. [80]. Many extensions have been proposed over recent years to improve the limitations of SPH methods. Becker and Teschner [51] propose to use the Tait pressure equation to reduce density fluctuations. Solenthaler and Pajarola [86] put forward a method to enforce incompressibility by a prediction-correction scheme to estimate particle pressures. Density fluctuations are iteratively propagated through particles until a targeted density is obtained. Their method to resolve pressures is a lot less expensive than solving a Poisson equation.

The projection step was also improved with a discretized form of the pressure Poisson equation [70]. This method was later extended to FLIP to combine the pressure projection step of the Implicit Incompressible SPH (IISPH) approach with the mass preservation and boundary handling of the FLIP method [58]. An exhaustive survey has been recently published [71]. Da et al. [60] recently introduced a surface-only liquid. They propose a way to store Lagrangian velocities directly into a mesh. Finally, Bender and Koschier [53] propose a two-solver approach (called DFSPH) to enforce low volume compression. They achieve a more stable and efficient SPH formulation by enforcing incompressibility on both position and velocity levels. Since our hybrid model needs a robust density correction step along the band, even though we developed our work on a standard SPH method, it proved convenient and intuitive to integrate DFSPH into our pipeline; therefore, we report most of our results on our implementation of DFSPH.

2.2.3. Multiscale Fluids

Several approaches propose to leverage the respective strengths of Lagrangian and Eulerian methods in order to model different scales of detail. Losasso et al. [77] use a particle level set (PLS) method to represent dense volumes of liquid and SPH particles for diffuse volumes. Diffuse particles allow reproducing different types of interaction between liquid and air (present in splashes, foam, whitewater, etc.). A similar approach using escape particles from the coarse simulation modeled by a PLS method has also been proposed by Lee et al. [75]. In their model, escape particles model small-scale details and are governed by an SPH scheme.

Other researchers focus their effort on such a hybrid model for air bubbles within a liquid [68]. They use a grid-based simulation to model large bodies of liquid and a Lagrangian method for the motion of small bubbles. Coupling between those two models allows the approach, through a split-and-merge process, to transfer contributions from small to large scale (and vice versa). This idea has been extended as well to simulate various speeds in simulations of smoke [64]. By representing high-speed smoke regions with particles, this approach effectively animates and models a large-scale scenario at low speed. Induced turbulence can also be preserved if this kind of approach is coupled with a localized hybrid method around obstacles [91]. An SPH-FLIP formulation is used to model local vorticity flow in those regions.

Raveendran et al. [81] introduce a clever way to exploit the strengths of both models (Eulerian and Lagrangian). They propose a hybrid SPH method that preserves its Lagrangian nature without generating grid artifacts. A divergence-free velocity field is enforced by resolving a Poisson equation on a coarse grid. In addition, hybrid models have provided ways to simulate fluids in real time by using adaptive grid structures [56]. Combining a Lagrangian method with a grid-based method is not the only way to model different scales in fluid simulations. Adaptive particle sizes can also be used to represent different scales of detail in a particle-based simulation [85]. A standard spatial grid using cubic cells is placed on top of a layer of tall cells in order to reduce computations. Moreover, preserving surface details using an Eulerian approach has been an active research topic in the last few years [54, 78]. Recently, Chentanez et al. [57] couple a grid-based method, a solver for Shallow Water Equations (SWE), and an SPH method. The SWE solver allows tracking of the fluid interface using a heightfield representation. Those equations are a simplified form of the Navier-Stokes equations, and provide a wave propagation model at the surface of a liquid. Surface elements are represented and resampled with particles following an SPH formulation. This idea has naturally been extended to the FLIP method [63]. Their band method resamples particles only within a narrow band near the surface. They also introduce a coupling scheme between particles and the grid to reduce energy fluctuations.

A portion of our hybrid model is inspired by Raveendran et al. [81]. Since our hybrid approach runs on the DFSPH model, it allows for significantly large timesteps with our Eulerian solver, while retaining stability. However, we chose to solve our Poisson equation by updating a zero level set that tracks our fluid interface instead of managing markers within the simulation domain. The level set method is also used to resample our particle surface band around the fluid surface. Although the intuition behind our band method is similar to the one proposed by Ferstl et al. [63], we concentrate its application exclusively on SPH methods (more like Chentanez et al. [57] do). Despite the latter similarities to our approach, we propose in comparison a non-homogeneous way (as presented by Desbrun [61]) to deal with density fluctuation and to constrain the particle band around the fluid interface. Our approach is very effective with large bodies of liquid while preserving SPH small-scale surface tension details where it matters.

Similarly to Solenthaler and Gross [85], we use a two-scale model to compute complex behaviors with SPH particles. However, instead of using feedback forces in-between the different resolutions, we use significantly heavier particles (larger masses and underneath the liquid surface) to correct the density while reducing substantially the requirement on the number of particles. Also, we use a generalized approach and simulate a narrow band of particles at the surface over the whole domain (as opposed to locally). Recently and in parallel with our work, Winchenbach et al. [89] presented an approach to allow infinite continuous adaptivity for incompressible SPH. Their scheme of mass redistribution allows smooth variation of the masses and sizes particles according to a distance to the surface of the fluid.

2.3. Method Overview

The core idea behind our hybrid method is to provide an effective way to handle the fine details with a particle-based surface band and the coarse volume of the liquid with an Eulerian model. We use a DFSPH model to represent the particle-based portion and to update the velocity field of the Eulerian grid underneath the liquid (see § 2.4). A first pressure estimation is computed from the grid and transferred to the SPH particles as a preconditioner when computing the divergence error (same as DFSPH). Similarly to Raveendran et al. [81], we compute an hybrid pressure term by combining values from both models (Eulerian and SPH) and allows preserving the Lagrangian behavior near the free surface. After computing the DFSPH steps, velocities from the particles are interpolated back to our velocity field and extrapolated afterward. Because we want to exclusively focus the DFSPH computations along the free surface, a few additional steps are performed to compute and update a narrow surface band of particles. We use a level set method to provide a proper way to track the liquid interface (as with the method of Gibou et al. [65]). The level set is advected using a semi-Lagrangian method, and is used to evaluate and resample our particle band thickness. For the Eulerian-DFSPH coupling, we leverage the idea of seeding larger-scale particles (which we call *fictitious*) at the transition between the particle band and the Eulerian grid. We use a non-homogeneous model to compute the density around that transition. An overview of our method is provided in Algorithm 3.

Furthermore, since our method deals with SPH band particles, fictitious particles, and grid cells, we introduce a notation to make algorithms and equations easier to understand. We use the subscripts b, f, and g to respectively identify the source of each quantity for band particles, fictitious particles, and grid cells. For instance, the quantities $\vec{x}_{f_i}$, $\vec{v}_{f_i}$, and $\vec{\rho}_{f_i}$ are respectively the position, velocity, and density of the i-th fictitious particle. We also use a compact notation to express quantities of multiple elements, e.g., $\rho_{(b,f)_i}$ as densities of the i-th band and fictitious particles. Naturally, the number of fictitious particles is always smaller than the number of band particles.

2.4. Hybrid Strategy

This part of our approach is inspired by Raveendran et al. [81]. Although we compute a hybrid pressure force as they do, ours is obtained by combining a first estimation from the Eulerian solver and the one from the divergence error step from DFSPH. The SPH particle initial states are used to initialize our level set at the first frame. In most of our scenarios, the grid velocity is initialized to zero at the first frame.

In order to adequately match the movement of the surface particles, a volume correction step is performed on the grid velocity field at each iteration to enforce a proper tracking of the liquid surface according to ϕ. An initial volume V_0 is computed from the simulation at

Algorithm 3: Simulation loop for our approach

```
Compute volume correction $(\nabla \cdot \vec{v}_g)_*$ (Eq. 2.4)
Apply correction to $\vec{v}_g$
Solve Poisson equation for $p_g$ using MICCG (§ 2.4)
Compute neighborhood for SPH and fictitious particles
for $\forall i$ do                                /* for every band and fictitious particle */
    Compute densities $\rho_{(b,f)_i}$ and factors $\alpha_{(b,f)_i}$
    Correct density error on $\vec{v}_{(b,f)_i}$ (DFSPH)
    Interpolate pressure $p_{g_c}$ of cell $c(\vec{x}_{(b,f)_i})$ to $p_{(b,f)_i}$
    Correct velocity $\vec{v}_{(b,f)_i}$ with $\nabla p_{(b,f)_i}$
    Compute densities $\rho_{(b,f)_i}$ and factors $\alpha_{(b,f)_i}$
    Correct divergence error on $\vec{v}_{(b,f)_i}$ (DFSPH)
    Compute $\triangle t$ according to CFL condition (Eq. 2.7)
    Integrate $\vec{v}_{(b,f)_i}(t + \triangle t)$ and $\vec{x}_{(b,f)_i}(t + \triangle t)$
for $\forall c$ do                                /* for every cell at $\phi < 0$ */
    Interpolate velocities $\vec{v}_{(b,f)}$ of particles to cell $c$ (Eq. 2.8)
    Extrapolate velocity $\vec{v}_{g_c}$
    Advect and update $\phi_{g_c}$
Resample particle surface band (§ 2.5.3)
Seed fictitious particles (§ 2.5.1)
Update non-homogeneous density (§ 2.5.2)
```

rest. Volume variation is obtained from the ratio between the current volume V_t and the one at its initial state V_0:

$$\triangle V = \frac{V_t - V_0}{V_0}. \tag{2.1}$$

The current volume V_t (as well as the volume at the initial state V_0) is computed with the Marching Cubes algorithm [76] on our level set to determine where liquid is present in the simulation domain. The cube configurations extracted (from each cell) allow estimating the space occupied by the liquid. The occupation ratio of a cell (cube) is equal to 1 if it is completely inside the liquid, and equal to 0 if it is completely outside. Otherwise, a cell spans the liquid interface with a ratio proportionally between 0 and 1. The volume variation (Equation 2.1) represents a difference between what we have and what we aim for. From this variation $\triangle V$, Kim et al. [73] compute a correction term c_V according to a loss or gain of volume:

$$c_V = -\left(\frac{k_p \triangle V - k_i \triangle V \triangle t}{\triangle V + 1}\right), \tag{2.2}$$

where terms k_p and k_i are obtained by performing an analysis on the volume change of a region under a constant divergence (see [73]) :

$$k_p = \frac{2.3}{\psi \triangle t}, \qquad k_i = \left(\frac{k_p}{2\xi}\right)^2. \tag{2.3}$$

Constants ψ and ξ are respectively fixed to 25 and 2 (refer to the paper of Kim et al. [73] for further details). Finally, the correction term c_V is used as a proportion control on the divergence field $\nabla \cdot \vec{v}_g$ to compensate for that variation in volume

$$(\nabla \cdot \vec{v}_g)_* = \frac{\nabla \cdot \vec{v}_g}{h} - c_V. \tag{2.4}$$

In Equation 2.4, $h = \frac{1}{n}$, where n is the grid resolution. Now that our divergence field $(\nabla \cdot \vec{v}_g)_*$ takes into account volume preservation, we have everything we need to solve for pressure on the grid. The linear system $\mathbf{Ax} = \mathbf{b}$ to solve is a Poisson equation, where $\mathbf{A}$ is a Laplacian defined by ϕ [50] used by the level set method, and $\mathbf{b}$ is the divergence field $\nabla \cdot \vec{v}_g$. To solve for $\mathbf{x}$ (i.e., for pressure) we use the Modified Incomplete Cholesky (MIC) Conjugate Gradient (CG) algorithm. The MIC algorithm is used as a sparse approximation for Cholesky factorization. It is very useful to speedup computation when applied as a preconditioner with the CG algorithm.

The pressure values obtained with our MICCG solver are transferred by simple trilinear interpolation from grid cells to SPH band and fictitious particles. As in the method of Raveendran et al. [81], we use those pressure values to compute the final pressure forces in our DFSPH model. We use the SPH formulation for pressure gradient to preserve the Lagrangian nature of our particles and where the current pressure (for particle i) and the ones from its neighbors (particles j) are transferred from the grid. The pressure force computed in the grid is used as a preconditioner in addition to the pressure values computed afterward with the special case equation of state ($\gamma = 1$ as with DFSPH). As presented by Bender and Koschier [53], we use the DFSPH equation to compute iteratively the divergence error $\vec{v}_i^{\,de}$:

$$\vec{v}_i^{\,de} = \triangle t \sum_j m_j \left(\frac{\kappa_i}{\rho_{(b,f)_i}} + \frac{\kappa_j}{\rho_{(b,f)_j}}\right) \nabla W_{ij}, \tag{2.5}$$

where κ is the stiffness constant also obtained while computing the divergence-free solver (see [53] for more details). As noted by Raveendran et al. [81], we save a few iterations (compared to the DFSPH method) by initializing $\vec{v}_i^{de}$ with our correction solved from the MICCG on the Eulerian grid. The hybrid pressure force is computed with the densities obtained from the first pressure solver (also called the density correction step in DFSPH). Then, the divergence correction step (second pressure solver) is performed on updated particle velocities. It allows our system to remain stable with large timesteps and without the use of a large stiffness constant (k as stated by Becker and Teschner [51]) with our SPH model.

We chose DFSPH as our particle-based model because of its low density fluctuation property and its stability. As for Bender and Koschier [53], the particle neighborhoods are updated using compact hashing. A maximum timestep is computed with respect to the Courant-Friedrichs-Lewy (CFL) condition [59, 79] with the form

$$\triangle t \leq h/c, \tag{2.6}$$

and it must satisfy the additional constraint

$$\triangle t \leq \min \frac{h}{|\vec{F}|}, \tag{2.7}$$

where c is the speed of sound (as used in thermodynamics [87]). Finally, a symplectic advection employing a Leap-frog scheme is used to integrate and obtain the updated velocity and position. The timestep $\triangle t$ is also used with our Eulerian solver. We update the grid velocities from these new values, as well as using an interpolation based on the position of particles (as with FLIP) in each cell.

$$\sum_{i=0}^{N-1} \vec{v}_{g_c}(\vec{x}_{(b,f)_i}) = \vec{v}_{g_c}(\vec{x}_{(b,f)_i}) + \text{trilerp}(\vec{v}_{(b,f)_i}). \tag{2.8}$$

We transfer the updated velocities $\vec{v}_{(b,f)}$ from the band and fictitious particles onto the grid faces. We also assign trilinearly weighted coefficients to each corner of cell c according to the barycentric coordinates of each band and fictitious particle within this cell. We then extrapolate the newly updated velocities in our grid to enforce consistency near the liquid interface. That interface (between liquid and air) is tracked where $\phi = 0$. The zero level set is essentially where the values in the associated distance field are equal to zero. We define our distance field to be]0,1] outside the liquid, and [−1,0[inside. It is relevant to mention that our level set uses the same grid as for SPH neighborhood. It allows us to effectively link particles from the surface band to the Eulerian part of the model. It is important to note that the level set is advected with our Eulerian grid in order to match the liquid interface if the resampling step within the band (§ 2.5.3) is done on highly turbulent and complex scenarios. Otherwise, it would have been far simpler to reconstruct the level set from the particles at each iteration.

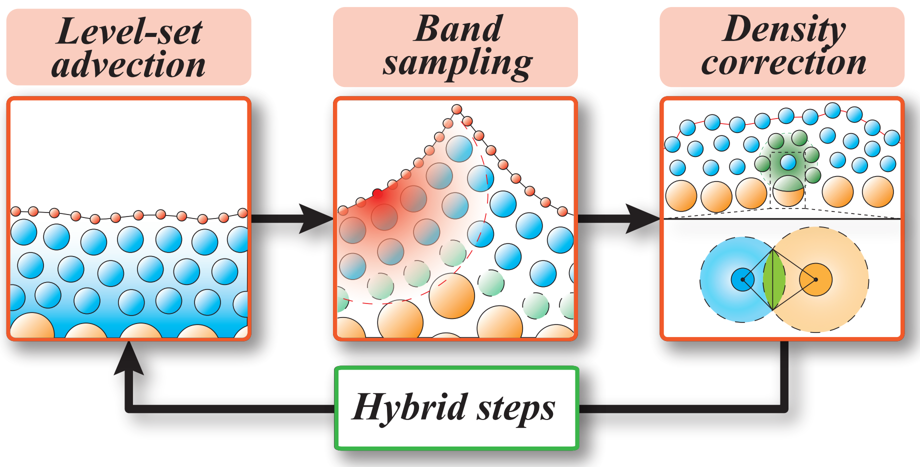


Figure 2.2. Schematic overview of our particle surface band algorithm.

2.5. Particle Surface Band

This section focuses on the additional steps required to maintain our SPH surface band along the liquid interface defined by the level set. As mentioned briefly in § 2.3, the particles are advected alongside the level set function. After each substep, the surface band needs to be resampled in order to preserve a consistent density among our SPH particles. Finally, a density correction step is performed to update the density among fictitious particles. These particles are used as probes inside the liquid to resample and correct the density of SPH particles along the surface band. A brief overview of these steps computed on the surface band is presented in Figure 2.2.

2.5.1. Lagrangian Nature

One of our main goals with the proposed approach is to preserve the Lagrangian nature (behavior and visual appearance) of an SPH method, without the necessity of handling particles with negligible visual contributions (i.e., deep inside the volume of liquid). Our hybrid method can be considered as two-way coupled by the manner it governs the coarse flow. Grid velocities are updated from the particles, and are then used to advect ϕ where the surface band is resampled. The fictitious particles can be considered as the coupling interface between the Lagrangian (SPH) and the Eulerian (grid) models of our approach. Because the adaptive timestep is computed from the SPH surface band particles, our level set (Eulerian part) does match fairly well. The SPH steps performed on the particles allow preserving the Lagrangian behavior and appearance. To achieve this, the steps of neighborhood search and density computation are crucial.

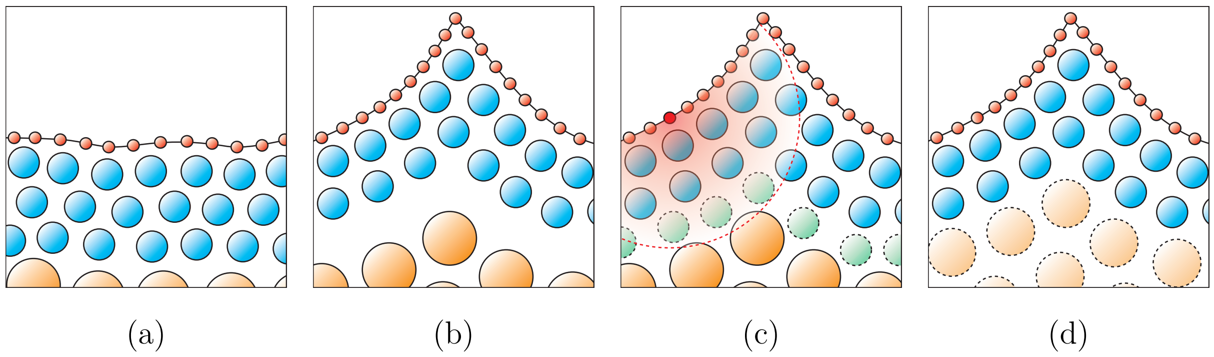
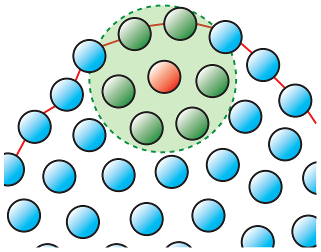


Figure 2.3. (a) Particles from the surface band (blue) are advected with our zero level set (red). (b) At each substep, a resampling must be performed to preserve a valid density between the surface band and the fictitious particles (orange). (c) The resampling process is performed using a Poisson sphere distribution [74] method based on every interface point defined by ϕ. (d) Resampling can also be performed among fictitious particles.

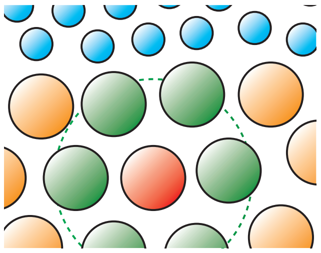
After the surface band resampling step (§ 2.5.3), we need to recover for any loss or gain of density within each significant volume of liquid. This is where the fictitious particles come into play. They apply a correction on density where there is neighborhood deficiency. These particles are seeded inside the liquid, so basically where $\phi < 0$. The resampling method is based on the grid resolution. By using the grid cells as resampling starting points, we simplify the search for these particles as neighbors. When we compute the density from these fictitious particles, the three following scenarios can arise.

2.5.1.1. *At the Band Interface*



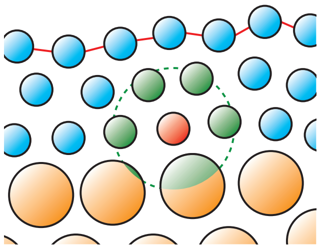
This is the simplest scenario, where there are only SPH particles (blue) that contribute to the current density (red). In other words, neighbors (green) of each particle are SPH particles. In that scenario, the standard SPH formulation for density computation is employed. However, note that some of those neighbor's densities might have been computed from other scenarios described below in §2.5.1.2 and § 2.5.1.3.

2.5.1.2. *Deep inside the Liquid without SPH Particles*



In that scenario, the density is computed exclusively from fictitious particles (orange). Similarly to density computation described in § 2.5.1.1, we use standard SPH formulation with significantly larger masses and smoothing radii. These density values are precomputed in case the nearby density of the SPH particles of the band is insufficient to preserve the stability of the simulation. More details on this specific scenario are discussed in § 2.5.2.

2.5.1.3. *Near the Transition of SPH and Fictitious Particles*



This is the most sensitive scenario, where SPH particles and fictitious particles are used together to contribute to density values. For those instances, we use a non-homogeneous density formulation to compute the density (as shown in § 2.5.2). By non-homogeneous, we mean that particles might have different masses. With fictitious particles, masses are significantly larger than the ones from SPH particles because of their proportional contribution (based on their masses and support radii) to the simulation. However, because we would like to prevent an expensive nonuniform pressure force computation (as done by Yan et al. [90]), the density needs to be similar to that of the traditional method (i.e., DFSPH in our case). Difficulties lie in the fact that the support radii (in addition to masses) are unequal among the particles. This matter is covered in detail in § 2.5.2.

Finally, while performing the volume computation step (§ 2.4), every cell is marked as part or not part of a significant volume of liquid. A volume is considered too small if it is below a certain threshold inside an isolated cell (without neighbor cells marked as liquid). Because of the grid resolution limitation (i.e., the goal is to keep memory requirements for the grid relatively small), our approach will not be able to resample a surface band to enclose these very small volumes. Nevertheless, these small volumes are simply treated by the traditional steps of the DFSPH method [53].

2.5.2. Non-homogeneous Density Model

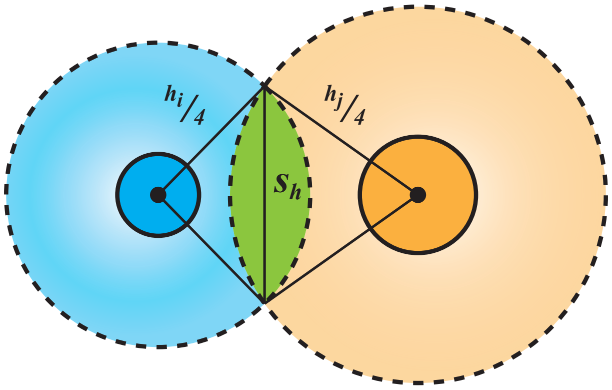


Figure 2.4. Example of overlapping support radii in 2D. In this case, the spherical caps are equal because they come from the same plane.

The fictitious particles are considered as neighbors as much as all other particles. They are stored in the same grid used for our level set ϕ, and so there is no additional treatment to search for them. The only validation needed is whether they lie inside the support radius of the current particle, which can be done at the neighborhood computation step. In order to take full advantage of the band approach, the number of fictitious particles must be significantly lower than in the reference high-resolution simulation. Densities of fictitious particles are computed like normal SPH particles. However, because of their relative masses m_{f_i} and support radii h_{f_i}, different kernel functions must be precomputed using these forms

$$m_{f_i} = \frac{3}{4}\pi\rho_0 r_i^3, \quad h_{f_i} = \epsilon\sqrt[3]{m_{f_i}/\rho_0}, \tag{2.9}$$

where ϵ is determined by the desired number of neighbors (as referred to by Desbrun [61] and revisited by Adams et al. [47] a few years later). The radius r_i of a particle is proportional to the particle spacing ω used at the initial state (i.e., $r_i = \lceil \frac{h_i}{\omega} \rceil$). As mentioned earlier, if there are enough neighbors around an SPH particle, we use the traditional way to compute density.

However, if there is a deficiency in the neighborhood, we must use a different formulation that combines SPH and fictitious neighbors. To prevent instabilities within the simulation, the size of a fictitious particle (proportional to its mass) must not overlap the one from an SPH particle (within the band). The initial particle spacing must be preserved at all times. When computing the density of an SPH particle (from the surface band) that has neighborhood deficiency, an estimation of the overlapping area between the two support radii must be performed (shown in Figure 2.4). Basically, we compute a ratio of that overlapping area (which is a volume between two spheres in 3D) and then multiply it by the density of the fictitious particle. The fictitious density contribution is expressed as

$$\rho^*_{f_i} = \alpha_{ij} \rho_{f_i}, \tag{2.10}$$

where α_{ij} is the overlapping volume ratio computed as

$$\alpha_{ij} = \frac{\pi s_{h_i}^2}{3} \left(3\frac{h_i}{4} - s_{h_i} \right) + \frac{\pi s_{h_j}^2}{3} \left(3\frac{h_j}{4} - s_{h_j} \right), \tag{2.11}$$

where s_{h_i} and s_{h_j} are respectively the spherical caps of the support radius influence (defined as a sphere in 3D) for the current particle i and his neighbor particle j. That density contribution is added as a corrective term to the SPH density of the current particle i. Even if this is an approximation of the neighboring density, the hybrid pressure computation prevents abrupt changes in density. It is important to note that in Equation 2.10, we consider the support radius volume to be equal to 1. However, in practice, the ratio α_{ij} must be divided by the current volume to be expressed as a normalized factor (i.e., between 0 and 1). Similarly to the band particles, the fictitious particle final velocities are computed as follows:

$$\vec{v}_{f_i} = \vec{v}_{f_i} - \triangle t \sum_j m_j \left(\frac{\kappa_i}{\rho^*_{f_i}} + \frac{\kappa_j}{\rho^*_{f_j}} \right) \nabla W_{ij}, \tag{2.12}$$

where $\kappa^v_{(i,j)}$ are the stiffness parameters for the current particle i and its neighboring particles j (see [52] for more details), and $\rho^*_{f_{(i,j)}}$ is computed with Equation 2.10.

2.5.3. Adaptive Band Resampling

The sampling step is performed along the surface band defined by ϕ (Figure 2.3). After the semi-Lagrangian advection step on ϕ, we perform a resampling among the SPH particles to enforce a minimum thickness at the liquid interface. We use a Poisson sphere distribution [74] pattern to reflect the current distribution of SPH particles (according to the initial

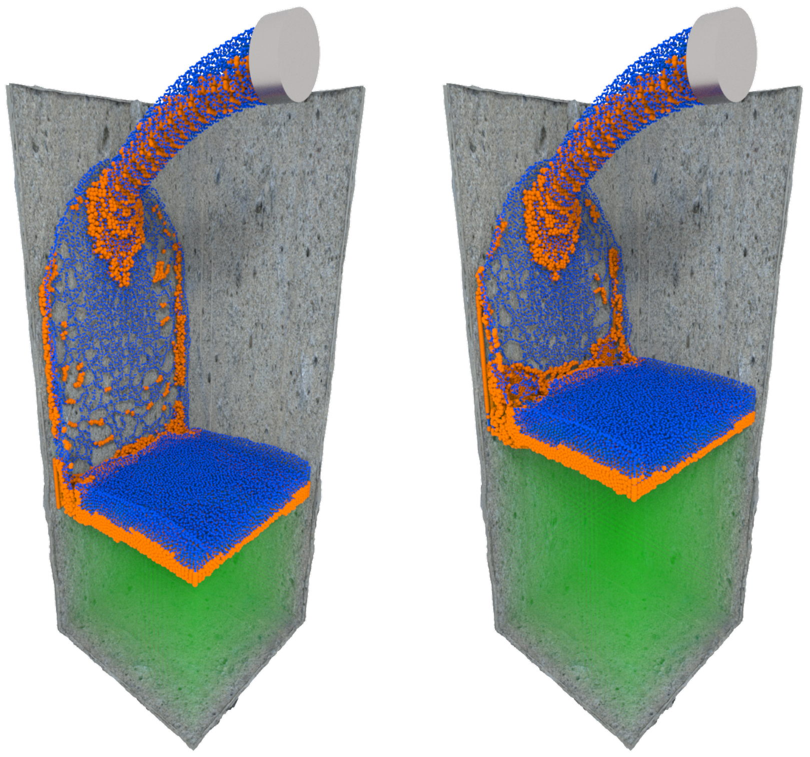


Figure 2.5. An example of pouring liquid in a container, with the number of particles increasing over time. The fictitious particles (orange) are seeded below the free surface and provide a correction step on densities to couple the Lagrangian (particle band) and the Eulerian solvers (grid cells in green).

particle separation). The region of interest to resample is near the transition between the particle surface band and the fictitious particles. As with [47], we use an additional resampling step to perform split and merge operations along the inside portion of the band to ensure that SPH densities are smooth across the interface and within this transition (as shown in Figures 2.3(c) and 2.3(d)). Moreover, in order to enforce a smooth transition at the surface band boundaries, we use a slightly larger SPH support radius to resample the particles along the surface band. The number of samples used to seed fictitious particles is currently a user-defined parameter. With all of our examples (except for the example of Figure 2.5 where only 3 to 4 samples were sufficient), we used 5 samples per cell. This number of samples offered experimentally a fair trade-off between stability and computation time. In some of our results and in our accompanying video sequences, some fictitious particles appear to lie outside the liquid surface. We extrapolate those samples in order to

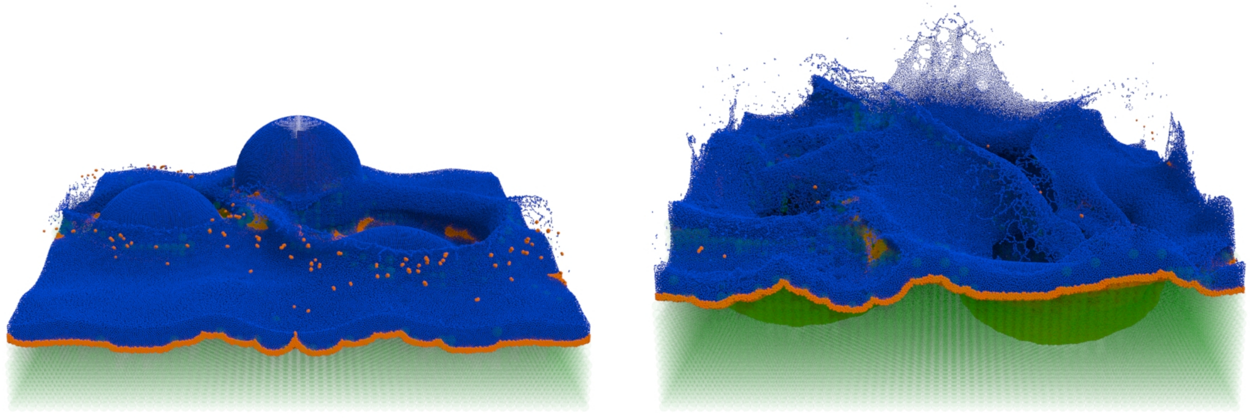


Figure 2.6. Three large drops falling in a container filled with the same liquid. The finer surface band particles (blue) lie on top of the inner fictitious particles (orange). The remaining portion under the free surface (green) is exclusively handled by our Eulerian solver, as illustrated by a regular grid of points.

ensure an adequate cover of the band when band particles are advected with ϕ. However, we do not consider fictitious particles if a band particle has a sufficient SPH neighborhood. In other words, at each timestep, several fictitious particles are never processed by our band algorithm. The neighborhood adequacy is determined by a minimum number of neighbors provided as a user-defined parameter (usually around 30-40, as stated by Monaghan [79]). Because particles at the free surface are known to suffer from particle deficiency, we only consider particles from a certain depth. This depth is basically at the boundary between the surface band and the fictitious particles inside our liquid.

2.6. Implementation and Results

Our method has been implemented on GPU with CUDA. The MICCG solver has been completely developed with CUDA, and most of the DFSPH implementation and our hybrid model (i.e., Eulerian level set) runs on GPU. For validation and comparison purposes, we also implemented every step of our approach on CPU, because some methods to which we compare ours (e.g., [81, 63]) were implemented exclusively on CPU. All CPU implementations for our method and the ones we compare to are run on a single CPU without multithreading. Our approach has been developed and is fully integrated as a geometry node with the development kit (HDK) in Houdini [84].

Method	**Example**		
	Fig. 2.1	Fig. 2.6	Fig. 2.5
WCSPH [51]	0.008	0.001	0.005
Hybrid-SPH [81]	2.1	2.2	3.1
PCISPH [86]	3.3	3.5	1.7
DFSPH [53]	3.9	4.1	2.8
Ours	4.1	4.2	2.8

Table 2.1. Comparisons of average timesteps used (in milliseconds) to simulate some of the presented examples.

We have managed to obtain significantly better computation times per step with our GPU implementation. For comparison, we simulated the scenario from Figure 2.6 on a GTX 1080 at 3 seconds per frame, leading to a 15× speedup. As stated earlier, we have managed to use a larger timestep compared to explicit SPH approaches. We were able to preserve the liquid stability with timesteps almost 2× larger than PCISPH and quite similar to DFSPH (as shown in Table 2.1). It is also worth mentioning that the orange particles shown in the results represent the presence of our fictitious particles seeded (using the same grid than for our distance field) beneath the surface (also visible in the video provided as supplementary material) and close to the free surface.

Method	**Example**		
	Fig. 2.1	Fig. 2.6	Fig. 2.5
WCSPH [51]	11339.3	4224.3	2826.1
Hybrid-SPH [81]	2050.2	763.8	542.6
PCISPH [53]	794.2	310.3	30.8
DFSPH [53]	110.5	41.7	5.2
NB-FLIP [63]	53.6	51.3	50.9
Ours	47.5	25.2	4.9

Table 2.2. Comparisons of computation times (in minutes) for all 500 frames between several state-of-the-art approaches that we implemented and ours.

In §2.6.1 to § 2.6.5, we present static images of scenarios simulated with our approach. Full animated sequences from these scenarios (and others) are available in the accompanying video. Finally, note that the bouncing behavior appearing in the first frames of some of the examples presented is inherited from the traditional SPH method.

2.6.1. Large Bodies of Liquid

Figure 2.6 shows a classic scenario simulating large drops of liquid falling in a container partially filled. This scenario is a good illustration of what our approach aims for: It is designed to perform better with large bodies of liquid, while we attempt to preserve as much as possible small-scale details at the surface (especially with splashes).

Table 2.2 illustrates the gap in timings between our approach and state-of-the-art methods proposed in recent years. Our method offers a considerable speedup compared to a full

Step	**Example (Fig.)**				
	2.1	2.6	2.8(a)	2.8(b)*	2.5
Volume correction	5.2%	6.3%	5.0%	6.3%	9.1%
MICCG solver	19.0%	18.8%	20.0%	18.8%	18.2%
Hybrid-DFSPH	31.0%	31.3%	30.0%	31.3%	27.3%
Advect & update ϕ	10.3%	9.4%	10.0%	12.5%	9.1%
Resample band	15.5%	15.6%	15.0%	12.5%	18.2%
Seed fictitious	10.3%	9.4%	10.0%	12.5%	9.1%
Fictitious density	8.6%	9.4%	10.0%	6.3%	9.1%
Total	5.7	3.1	1.8	1.5	1.1

Table 2.3. Average computation time breakdown per step (percentage values are relative to the total time in seconds) for some of our examples. *This scenario (Figure 2.8(b)) is a good example that shows the limitation of our approach with small volumes and thin layers of liquid.

resolution hybrid model [81] and is very similar in computation times to the narrow band FLIP [63]. Although our approach does not achieve real-time performances as Chentanez et al. [57], we manage to offer rather competitive offline computation times while retaining the Lagrangian nature of the SPH model without altering its behavior through the coupling process. Our hybrid method preserves most of the small-scale details offered by the state-of-the-art SPH method. We took advantage of a similar hybrid approach from Raveendran et al. [81] to estimate pressure forces from a hybrid model and by focusing our efforts on a band-oriented method. It is also important to mention that the *Hybrid-DFSPH* computation times from Table 2.3 include the steps of density and divergence correction. As shown in Table 2.3, the steps required by our surface band (resampling and non-homogeneous density) do not require much additional computation time compared to processing a full resolution SPH method, with its neighborhood and high pressure force. Another interesting example is shown in Figure 2.5. In that scenario of pouring liquid, the number of particles is always increasing over time. As expected, the larger the volume of liquid, the better our approach performs. However, such an example is difficult to properly benchmark because of this variation in complexity (see Table 2.3). It is a very good trade-off because the SPH method typically performs well with fewer particles, and our extensions offer excellent performance when processing large bodies of liquid.

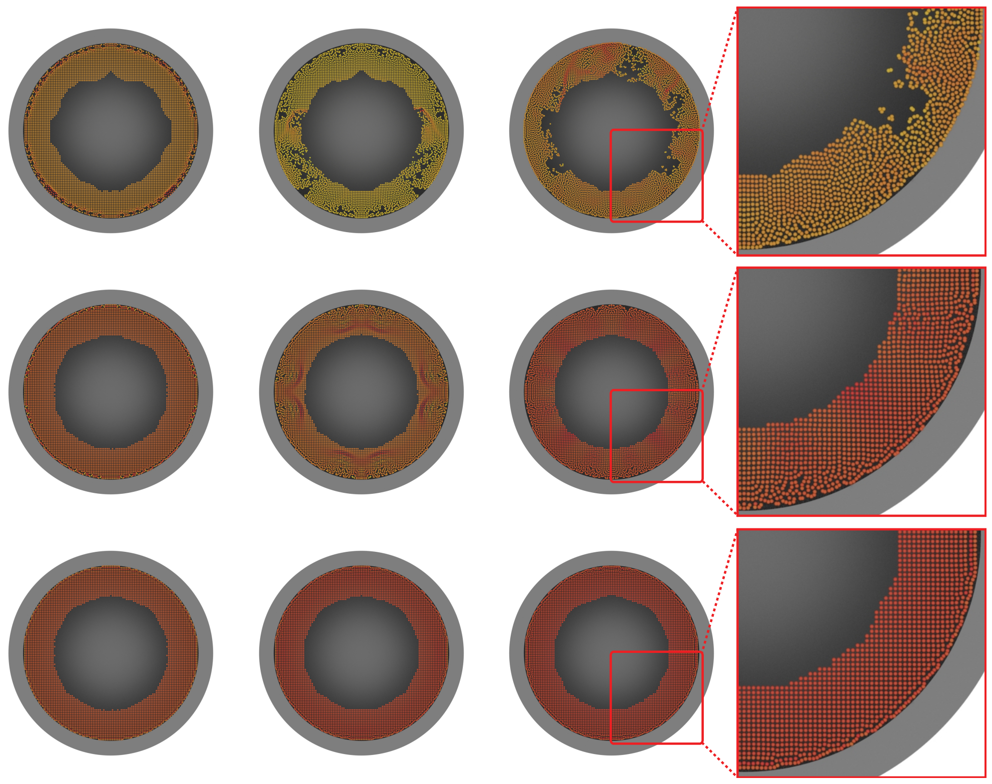


Figure 2.7. Influence of the resampling of fictitious particles over the density correction (illustrating the stability) along the surface band, especially around the outer interface. The boundaries are handled by ghost solid particles (grey regions) and the fictitious particles are resampled in the empty space inside the sphere. The columns of the figure, from left to right, represent different times in the simulation, and the rows from top to bottom represent the number of fictitious particles per cell (set respectively to 1, 3, and 5).

2.6.2. Density and Divergence Correction

As mentioned earlier, applying density correction within a band-oriented SPH method is not trivial. An isolated scenario has been used to validate the density correction step along the particle surface band. The setup for this scenario simulates a constrained sphere of particles without gravitational forces. The velocities are updated using artificial viscosity forces and corrected at the pressure projection step. We used ghost air particles (represented by the grey boundaries in Figure 2.7) introduced by Schechter and Bridson [83] to spatially constrain the liquid interface over time. Figure 2.7 illustrates different numbers of fictitious particles used to adjust density along the surface band (especially along the interior interface). In most of our examples, we use around 5 fictitious particles per cell. However, this number may differ depending on the type of scenario. The most sensitive region is clearly along the surface band inside the liquid (the inner interface). This region needs to be resampled every step with SPH particles (after advection). As shown in Figure 2.7, most of our examples may become unstable after a few steps in presence of too few (i.e., below 3) fictitious particles per cell.

An intuitive way to understand the relation between the number of samples (i.e., number of fictitious particles) and the stability of the band is by observing the grid cells in those regions. Since we use a hybrid pressure term as a first approximation for the DFSPH solvers, the fewer fictitious particles we have in these coupling regions and the more biased this approximation will be. However, based on our multiple experiments, most of the time using a small number of fictitious particles along the SPH particle band will result in a loss of small-scale details.

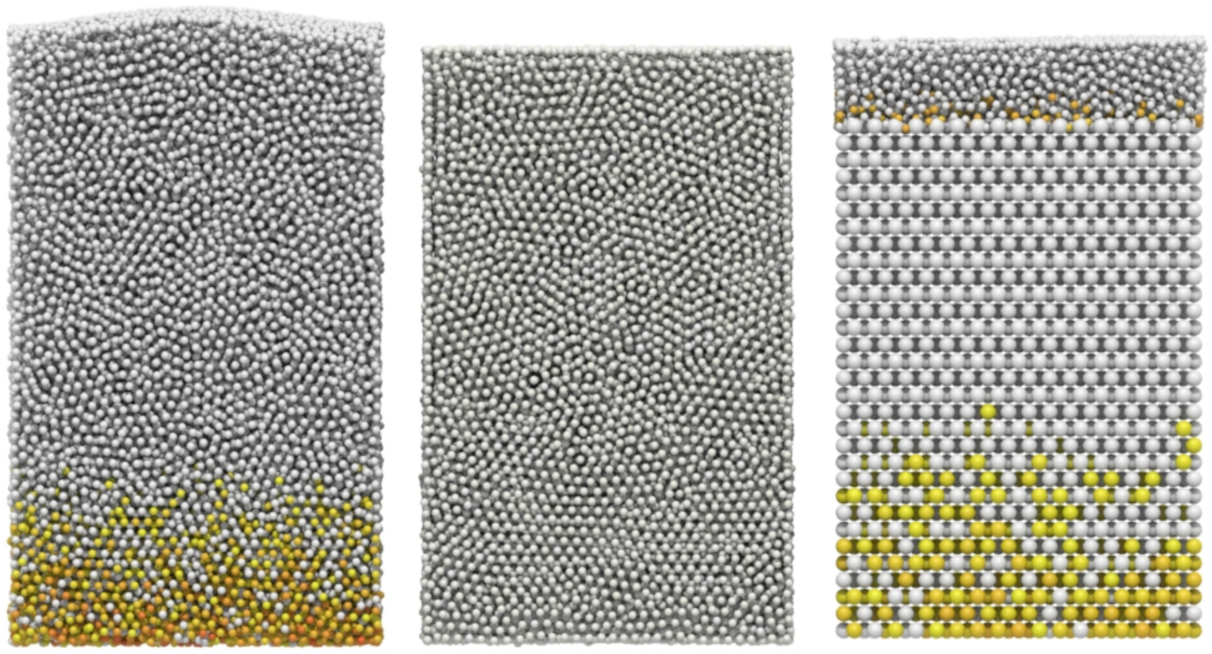


Figure 2.9. Columns of stabilizing liquids (80k particles). These color-coded examples compare the divergence errors respectively between (left) PCISPH, (center) DFSPH, and (right) our method (grid cells, band and fictitious particles). The errors are color coded from white (no error) to red (from yellow to red in HSV color space).

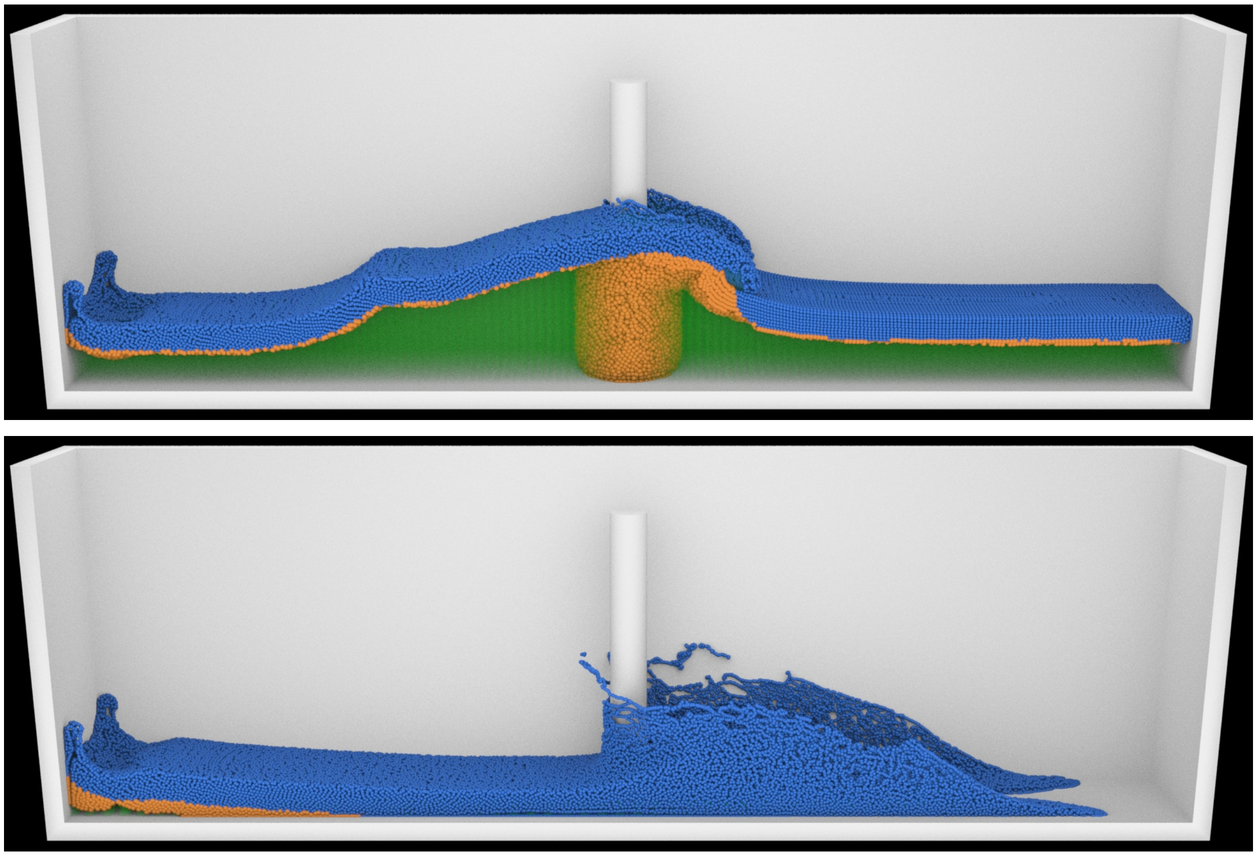


Figure 2.8. (top) An example where we use ghost solid particles to interact as a static obstacle with our particle surface band. (bottom) An example where our approach is unable to capture the thin layers. The volume of liquid is too small to be considered for band resampling and is therefore only handled by the DFSPH solver.

Along the same line of ideas, the density computed at the boundaries of the particle-band (between the inside of the liquid and the band) is essential to the stability of the simulation. The pressure forces computed (with these densities) in these areas are mostly responsible for the size of the timestep used and therefore for the computation time per iteration before convergence. An insufficient number of fictitious particles can create instabilities (even with a short timestep). Otherwise, an erroneous density can also result in loss of fine surface details. We used the same scenario (fluid hitting a pillar with 80k particles) than Bender and Koschier [52] to compare our resulting divergence error. Figure 2.9 shows the divergence errors compared to other state-of-the-art methods. As shown, PCISPH presents some divergence errors especially close to the boundaries (deeper inside the liquid). On the other hand

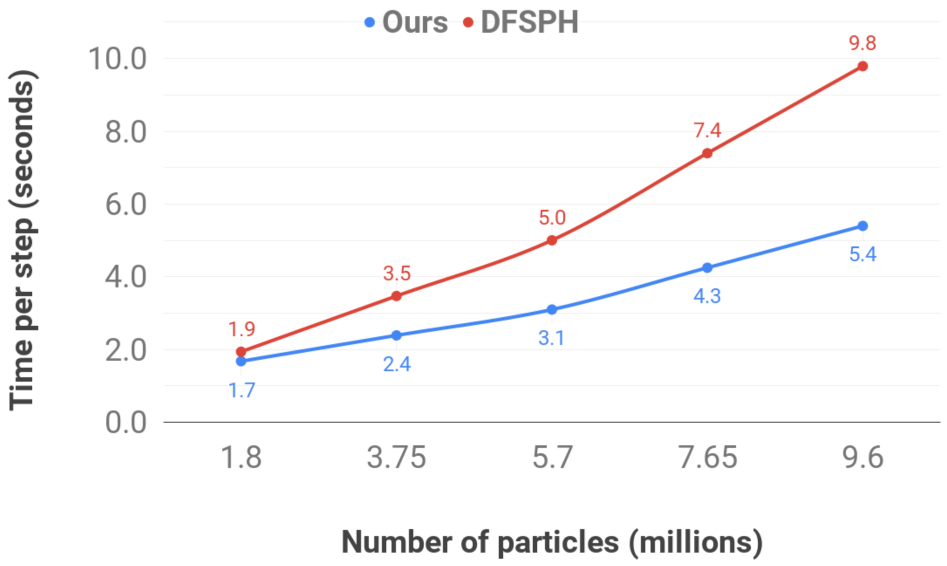


Figure 2.10. Performance comparison between DFSPH and ours on example Fig. 2.6.

and as expected, DFSPH shows almost no significant divergence errors within the simulation domain. Our method introduces a maximum local divergence error of 11.4 s^{-1} in different regions within our hybrid model. As exposed in Figure 2.9, our divergence errors mostly occur deep in the liquid since the density estimation is solely performed on grid cells. This divergence error is larger than for DFSPH (around 3 s^{-1} for that same scenario), but significantly lower than the one occurring with PCISPH (around 73 s^{-1}). The error seems to increase within a thin layer nearby the transition in-between the surface band and the fictitious particles. This miscalculation is essentially caused by the way the density is computed for these regions. The correction made from the overlapping support radii step on non-homogeneous density model (§ 2.5.2) reduces the related error but still, it remains present. An accumulated divergence error also lies underneath the surface since the fictitious particle densities are computed from fewer neighborhood contributions with much larger masses.

Furthermore, the main advantage of our approach is clearly exposed as we increase resolution (i.e., number of particles) for a specific example. As shown in Figure 2.10, although our method is slower for small numbers of particles compared to pure DFSPH, it shows better speedup factors as resolution increases. Similarly to the issue explained in Figure 2.8, our hybrid framework is slightly more demanding for small volumes of liquid and coarse

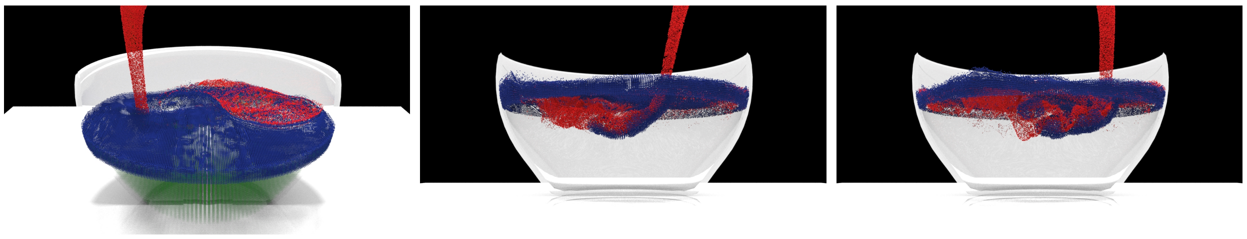


Figure 2.11. Example of mixing different density/viscosity liquids in a bowl (e.g., water in blue and oil in red). Our method updates a second particle band underneath the liquid to handle non-homogeneous liquid-liquid interactions. The green portion below the surface (selected grid cells on the left image) is handled by the Eulerian solver and coupled with the fictitious particles resampled along both surface bands.

discretizations since our hybrid model requires additional steps to approximate the deep portions and to handle the coupling of both models (i.e., grid and particles).

2.6.3. Flows with Obstacles

Our approach is also compatible with ghost solid particles to support static and dynamic boundaries (such as obstacles). Figure 2.8(a) shows an example where our approach successfully handles a static obstacle (unlike Figure 2.8(b), with not enough liquid). With the container partially filled with liquid, we can define a particle surface band within a reasonable volume. The level set does not consider the ghost particles and allows us to extend our surface band around the obstacle.

2.6.4. Density-contrast Liquid-liquid Interactions

Our method has also proven to be able to handle several narrow bands within the same simulation domain. In the example shown in Figure 2.11, we use our approach to model the interaction of mixing two different density (and viscosity) liquids. Each liquid of different density is represented by a single band of particles (blue for water and red for oil). In order to adequately follow the interface of each liquid, distinct level sets have to be updated. Moreover, some band particles are discarded underneath the liquid where both level sets are overlapping. We usually keep ϕ (and its associated band) from the densest liquid. In this scenario, we used a grid pattern to initialize the particles which explain the visible pattern

Example : Grid res.	# particles			Ratio (%)	
	Full res.	Band	Fictitious	$\frac{Band}{SPH}$	$\frac{Fict.}{Band}$
Fig. 2.1 : $96 \times 96 \times 96$	10.2M	2.1M	310k	20.6	14.8
Fig. 2.11 : $64 \times 64 \times 64$	2.5M	575k	123k	23	21.4
Fig. 2.6 : $64 \times 64 \times 64$	5.7M	800k	175k	14.1	21.9
Fig. 2.8(a) : $32 \times 32 \times 64$	200k	75k	17.5k	37.5	23.3
Fig. 2.8(b)* : $32 \times 32 \times 64$	100k	90k	<5k	90	–
Fig. 2.5 : $32 \times 40 \times 32$	–	12.5k	10.5k	–	–

Table 2.4. Parameters and statistics related to our simulation examples. The number of particles used for the band and the fictitious is averaged over all the simulation frames.

(compare to the red particles emitted). We used the same initialization method for the initial state used with Figure 2.6.

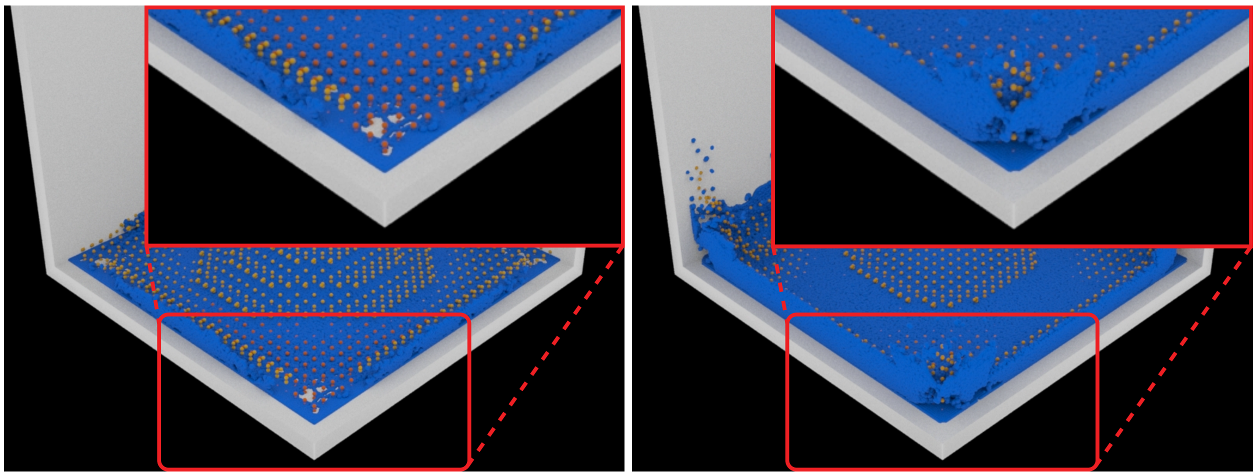


Figure 2.12. Instabilities can occur when our level set is not able to capture small features in the form of too-thin layers.

2.6.5. Limitations with Thin Layers

Our approach is able to process small-scale details within the limits of the resolution of our level set (defined inside a grid). Because of this grid resolution, a detached volume of liquid must have an above-threshold volume to be considered. The grid resolution must be kept fairly low in order to achieve reasonable computation times per step, but it must be high enough to follow a much more deformed interface of the liquid. Table 2.4 shows grid resolutions used in our examples. In some examples, there is so much high-frequency motion that our level set is unable to match the actual interface. We could increase the grid resolution, but at the cost of increasing the number of fictitious particles of smaller masses in order to preserve our density. It is also interesting to mention that the example shown in Figure 2.8(b) is a good illustration of when our approach fails to deliver a performance gain (compared to PCISPH). As exposed in Table 2.3, the average computation time per step gets larger than using exclusively a hybrid model instead (such as [81]). The additional framework required by our approach makes each step very expensive in order to preserve an adequate surface band (which is mostly the whole simulation). We could overcome this limitation by using an adaptive structure (e.g., k-d tree) for small volumes of liquid, but we have not implemented it so far.

Another limitation with thin layers of liquid is related to a specific case of surface tension. A typical and perfect example of surface tension is to let a single drop of liquid fall on the ground. The inability to track and preserve a coherent interface makes it really difficult to resample around the surface band. Moreover, in that specific example, it causes instability in the corners of the container (as shown in Figure 2.12). The reason for this is the deficiency in these neighborhoods when hybrid pressure forces are computed. It accumulates errors based on few fictitious particles and almost no SPH neighbors. Switching to a standard SPH method resolves those instabilities when small volumes are detected. Consequently, most of our examples use a traditional SPH method locally in areas where there are thin layers of liquid (or insufficient density information) at the free surface. A mesh-based method (e.g., [60]) would be much more adequate for this specific example. Also as exposed by He et al. [67], surface tension with SPH methods is still an active research topic.

2.7. Conclusion and Future Work

Our approach is very effective with a wide range of scenarios featuring small-scale details in large bodies of liquid. The band-only method is an increasing trend in particle-based fluid simulation for computer graphics because it allows focusing efforts mainly on the visible parts of an animated fluid. Our approach offers an Eulerian-DFSPH scheme along a non-homogeneous density model to greatly reduce the particle count requirement in comparison to a full-resolution SPH simulation. We achieve a considerable speedup while preserving the expected behavior and realistic level of detail of an SPH model.

We can witness some losses in volume when small detached portions of liquid are removed after the resampling step on the surface band. As future work, it would be possible to treat these small detached volumes as diffuse particles [77]. To some extent, it would be also interesting to push the limits of the proposed approach to a fully adaptive model with multiple scales of particles (similar to [69] and [89]). Finally, we plan to use our band-oriented approach for procedural surface details. We could improve local regions from our particle band in order to add induced turbulence, splashes, and other diffuse behaviors such as liquid-air mixtures.

Acknowledgments

This work was supported and funded by Mitacs, Prompt, NSERC, Digital District Canada (formerly Mokko Studio), and the University of Montreal. We would like to thank Digital District for providing resources and insightful comments during our research. We would also like to thank Side Effects Software for providing us with Houdini educational licenses.

References

[46] Mridul Aanjaneya, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Trans. on Graphics*, 36(4):Art. 140, 2017.

[47] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J Guibas. Adaptively sampled particle fluids. *ACM Trans. on Graphics*, 26(3):Art. 48, 2007.

[48] Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Trans. on Visualization and Computer Graphics*, 18(8):1202–1214, 2012.

[49] Ryoichi Ando, Nils Thürey, and Chris Wojtan. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. on Graphics*, 32(4):Art. 103, 2013.

[50] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. on Graphics*, 26(3):Art. 100, 2007.

[51] Markus Becker and Matthias Teschner. Weakly compressible SPH for free surface flows. In *Proc. Symposium on Computer Animation*, pages 209–217. ACM SIGGRAPH/Eurographics, 2007.

[52] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 147–155. ACM, 2015.

[53] Jan Bender and Dan Koschier. Divergence-free SPH for incompressible and viscous fluids. *IEEE Trans. on Visualization and Computer Graphics*, 23(3):1193–1206, 2017.

[54] Morten Bojsen-Hansen and Chris Wojtan. Liquid surface tracking with error compensation. *ACM Trans. on Graphics*, 32(4):Art. 68, 2013.

[55] J.U. Brackbill and H.M. Ruppel. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics*, 65(2):314–343, 1986.

[56] Nuttapong Chentanez and Matthias Müller. Real-time Eulerian water simulation using a restricted tall cell grid. *ACM Trans. on Graphics*, 30(4):Art. 82, 2011.

[57] Nuttapong Chentanez, Matthias Müller, and Tae-Yong Kim. Coupling 3D Eulerian, heightfield and particle methods for interactive simulation of large scale liquid phenomena. *IEEE Trans. on Visualization and Computer Graphics*, 21(10):1116–1128, 2015.

[58] Jens Cornelis, Markus Ihmsen, Andreas Peer, and Matthias Teschner. IISPH-FLIP for incompressible fluids. *Computer Graphics Forum*, 33(2):255–262, 2014.

[59] Richard Courant, Kurt Friedrichs, and Hans Lewy. On the partial difference equations of mathematical physics. *IBM Journal*, 11(2):215–234, 1967.

[60] Fang Da, David Hahn, Christopher Batty, Chris Wojtan, and Eitan Grinspun. Surface-only liquids. *ACM Trans. on Graphics*, 35(4):Art. 78, 2016.

[61] Mathieu Desbrun. *Space-time adaptive simulation of highly deformable substances.* PhD thesis, INRIA, 1999.

[62] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation'96*, pages 61–76. Springer, 1996.

[63] Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. Narrow band FLIP for liquid simulations. *Computer Graphics Forum*, 35(2):225–232, 2016.

[64] Yue Gao, Chen-Feng Li, Shi-Min Hu, and Brian A Barsky. Simulating gaseous fluids with low and high speeds. *Computer Graphics Forum*, 28(7):1845–1852, 2009.

[65] Frederic Gibou, Ronald P Fedkiw, Li-Tien Cheng, and Myungjoo Kang. A second-order-accurate symmetric discretization of the Poisson equation on irregular domains. *Journal of Computational Physics*, 176(1):205–227, 2002.

[66] Robert A. Gingold and Joseph J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181(3):375–389, 1977.

[67] Xiaowei He, Huamin Wang, Fengjun Zhang, Hongan Wang, Guoping Wang, and Kun Zhou. Robust simulation of sparsely sampled thin features in SPH-based free surface flows. *ACM Trans. on Graphics*, 34(1):Art. 7, 2014.

[68] Jeong-Mo Hong, Ho-Young Lee, Jong-Chul Yoon, and Chang-Hun Kim. Bubbles alive. *ACM Trans. on Graphics*, 27(3):Art. 48, 2008.

[69] Christopher Jon Horvath and Barbara Solenthaler. Mass preserving multi-scale SPH. Technical Report 13-04, Pixar Animation Studios, 2013.

[70] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible SPH. *IEEE Trans. on Visualization and Computer Graphics*, 20(3):426–435, 2014.

[71] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH fluids in computer graphics. In *STAR*. Eurographics, 2014.

[72] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Trans. on Graphics*, 34(4):Art. 51, 2015.

[73] Byungmoon Kim, Yingjie Liu, Ignacio Llamas, Xiangmin Jiao, and Jarek Rossignac. Simulation of bubbles in foam with the volume control method. *ACM Trans. on Graphics*, 26(3):Art. 98, 2007.

[74] Ares Lagae and Philip Dutré. Poisson sphere distributions. In *Vision, Modeling, and Visualization*, pages 373–379, 2006.

[75] Ho-Young Lee, Jeong-Mo Hong, and Chang-Hun Kim. Interchangeable SPH and level set method in multiphase fluids. *The Visual Computer*, 25(5):713–718, 2009.

[76] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH*, pages 163–169. ACM, 1987.

[77] Frank Losasso, Jerry Talton, Nipun Kwatra, and Ronald Fedkiw. Two-way coupled SPH and particle level set fluid simulation. *IEEE Trans. on Visualization and Computer Graphics*, 14(4):797–804, 2008.

[78] Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim, and Derek Nowrouzezahrai. Surface turbulence for particle-based liquid simulations. *ACM Trans. on Graphics*, 34(6):Art. 202, 2015.

[79] Joe J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574, 1992.

[80] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proc. Symposium on Computer Animation*, pages 154–159. ACM SIGGRAPH/Eurographics, 2003.

[81] Karthik Raveendran, Chris Wojtan, and Greg Turk. Hybrid smoothed particle hydrodynamics. In *Proc. Symposium on Computer Animation*, pages 33–42. ACM SIGGRAPH/Eurographics, 2011.

[82] Bruno Roy and Pierre Poulin. A hybrid Eulerian-DFSPH scheme for efficient surface band liquid simulation. *Computers & Graphics*, 77:194–204, 2018.

[83] Hagit Schechter and Robert Bridson. Ghost SPH for animating water. *ACM Trans. on Graphics*, 31(4):Art. 61, 2012.

[84] SideFX. Houdini (version 15.5.717). `https://www.sidefx.com/products/houdini-fx/`, 2016.

[85] Barbara Solenthaler and Markus Gross. Two-scale particle simulation. *ACM Trans. on Graphics*, 30(4):Art. 81, 2011.

[86] Barbara Solenthaler and Renato Pajarola. Predictive-corrective incompressible SPH. *ACM Trans. on Graphics*, 28(3):Art. 40, 2009.

[87] John Lighton Synge. *The Relativistic Gas*, volume 32. North-Holland Amsterdam, 1957.

[88] Kiwon Um, Seungho Baek, and JungHyun Han. Advanced hybrid particle-grid method with sub-grid particle correction. *Computer Graphics Forum*, 33(7):209–218, 2014.

[89] Rene Winchenbach, Hendrik Hochstetter, and Andreas Kolb. Infinite continuous adaptivity for incompressible SPH. *ACM Trans. on Graphics*, 36(4):Art. 102, 2017.

[90] He Yan, Zhangye Wang, Jian He, Xi Chen, Changbo Wang, and Qunsheng Peng. Real-time fluid simulation with adaptive SPH. *Computer Animation and Virtual Worlds*, 20(2-3):417–426, 2009.

[91] Bo Zhu, Xubo Yang, and Ye Fan. Creating and preserving vortical details in SPH fluid. *Computer Graphics Forum*, 29(7):2207–2214, 2010.

[92] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. on Graphics*, 24(3):965–972, 2005.

Chapter 3

Particle Upsampling as a Flexible Post-Processing Approach to Increase Details in Animations of Splashing Liquids

This second paper presents an approach for enriching existing animations of splashing liquids. In order to achieve this, we construct and upsample a splash volume to induce splash dynamics. Ballistic particles are used to simulate diffuse dynamics and infer surface tension bevahior with artificial forces. In addition, we use an implicit wave model to couple interactions between splashing particles and the existing coarse volume of liquid.

Publication

This paper [130] was published in Computers & Graphics, Volume 88, in May 2020. It has been reformatted for this thesis. Roy was involved in the formulation and numerical analysis of this approach. He was also responsible for the design, implementation, and evaluation of the approach. Poulin supervised each stage of the research project. The first draft of the published paper was written by Roy and improved by Paquette and Poulin prior and during the submission process.

Particle Upsampling as a Flexible Post-Processing Approach to Increase Details in Animations of Splashing Liquids

BRUNO ROY, *Université de Montréal*
ERIC PAQUETTE, *École de technologie supérieure*
PIERRE POULIN, *Université de Montréal*

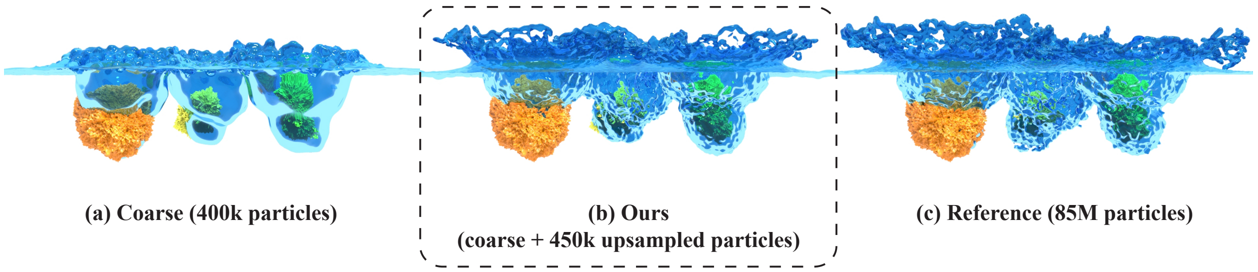

(a) Coarse (400k particles)
(b) Ours
(coarse + 450k upsampled particles)
(c) Reference (85M particles)

Figure 3.1. Induced splash dynamics is generated (b) on top of a coarse FLIP liquid provided as input (a). Our approach is able to reproduce localized realistic splashes comparable to a high-resolution (about 100× more particles) reference (c) with solely 450k upsampled particles seeded within a small portion of the domain.

Abstract

Realistic and detailed liquid splashes require costly fine-scale discretization. We present an efficient post-processing approach for particle-based methods to locally improve the behavior of splashes on coarser liquids. Our method first computes a splash volume over time from the intersections between an identified upsampling volume and colliding volumes. We then upsample particles inside cells of the splash volume; these cells are pre-computed using a likelihood score based on criteria favoring emerging particles. In addition to the advection scheme, enhanced realism is achieved by applying a localized artificial pressure on upsampled particles in order to mimic surface tension in critical regions of splashes. Finally, we propagate waves using a novel implicit model that couples the impact of upsampled particles on the coarser liquid by updating the velocity field at these locations. Our implicit wave model can produce detailed swirls by solely applying velocity updates directly on the underlying particles from the coarse liquid, and prevents from using a high density of surface points. As a result, our approach can generate localized and parameterizable high-resolution splashes from solid-liquid and liquid-liquid interactions, and thus can simulate a wide range of unique and customizable splashes on top of an animated coarse liquid.

Keywords: Splash Modeling, Post-processing Approach, Procedural Method, Multi-scale, Multi-resolution, FLIP

3.1. Introduction

In liquid simulation for visual effects, the discretization of the model is key to extract important features from complex examples, such as splashes and highly turbulent flows. This implies that in a particle-based liquid simulation, the number of particles must be strongly increased to bring out the fine details desired for the final outcome. With a uniform density of particles, most of the particles do not contribute to the desired and expected behavior displayed in the final rendered image, as they are either submerged below the surface or remain almost motionless throughout the simulation.

In this paper, we propose a flexible post-processing approach to improve splash modeling on top of a lower-resolution FLIP liquid surface. A subset of the input liquid volume is extracted from an input FLIP simulation. The extracted volume, so-called *splash volume*, is generated using the solid-liquid or liquid-liquid interactions occurring during the input animation. This volume is then augmented with a denser set of additional particles advected through the input velocity field. *Tracer* particles are seeded among the upsampled particles to control the behavior of the denser splash volume introduced by our approach. These tracers exert artificial forces, allowing the simulation of a surface-tension effect similar to applying an artificial pressure force. The collisions between airborne upsampled particles and the surface of the input coarse liquid are used as impact sources that inject energy into a surface-based linear wave model.

We present a novel and comprehensive splash modeling approach for particle-based liquids. Our main contributions can be summarized as the introduction of:

- A volume-based method to determine and upsample the splash zone for liquid-liquid and solid-liquid interactions.
- Localized tracers to induce diffuse and surface-tension behaviors on upsampled particles.

- A sparse implicit linear wave formulation to handle interactions between the coarse input and the upsampled particles at impact locations.
- A trade-off between realistic and artistic splash effects.
- An interactive splash editing approach (once pre-computation steps are performed).

3.2. Related Work

Over the years, several solutions to simulate realistic and detailed fluids have been proposed for purely Lagrangian methods, such as explicit and implicit SPH [139, 111, 98], and for hybrid methods, such as FLIP [152, 99, 95, 107, 131]. We refer readers to a comprehensive survey [112] and a book [101] describing in detail these methods. Procedural methods have also proven to be very efficient to enrich an input coarse fluid with fine details while offering a more practical artistic control. In the following, we focus primarily on topics more closely related to our work.

3.2.1. Upsampling Methods and Adaptive Models

Several approaches have explored different ways to increase the apparent spatial resolution of a coarse fluid simulation. In the last decade, the challenge of increasing the apparent resolution of fluid simulations has been addressed through many variants of sophisticated upsampling methods and spatial adaptive simulation models.

3.2.1.1. *Upsampling Methods*

These methods are able to effectively improve the high-frequency details while preserving the low-frequency behavior of the fluid, especially for smoke simulations [117, 127, 133, 129]. Although we focus solely on increasing the resolution of localized splash behaviors in liquids, our work shares this same goal. Few attempts to apply this goal to liquids were made for both Eulerian [127] and Lagrangian [149, 135] methods. Instead of upscaling the input velocity field, our method provides a realistic way to upsample localized portions of the unmodified coarse simulation data (particles and velocity field).

Some researchers have considered adding sub-scale quantities (e.g., curl noise) to existing coarse simulations [117, 127], while others have coupled these quantities to the Navier-Stokes equations [133]. In all these techniques, the quantities are extrapolated into the obstacles (geometric models) to prevent artifacts around them, but at the cost of significantly less realistic behaviors compared to high-resolution simulations. Selle et al. [134] proposed an approach to introduce particle-based vorticity to grid-based methods. Their work was extended by Pfaff et al. [129] to handle dynamic obstacles. In contrast to adding sub-scale quantities to existing simulations, other researchers have introduced multi-scale and adaptive formulations for particle-based methods that extract different ranges of details within a simulation domain [137, 97, 95, 96].

Kim et al. [116] introduced a novel Eulerian approach to increase the apparent spatial resolution of an existing liquid simulation. As they noted, increasing the free-surface details has shown to be a valid solution to the up-scaling problem on liquids, since the tangential velocity (and associated turbulence) is loosely coupled to the fluid velocity field. Their work was extended by Mercier et al. [122] to propose a coupled scheme using a high-resolution wave simulation while preserving the entire frequency spectrum of the input coarse simulation. For more details, we refer readers to the comprehensive survey by Thuerey et al. [142] on recent fluid up-scaling methods. These methods are unable to generate small-scale details and are limited to increase the actual apparent resolution. We offer a complementary and efficient way to recreate the missing small-scale details at sudden disturbance locations occurring with coarse liquids.

3.2.1.2. *Spatial Adaptive Models*

Even though our approach acts as a post-processing method, it is closely related to the essence of several spatial adaptive simulation models of the state of the art. Adjusting the spatial resolution of simulation models has been firstly introduced for fluid simulation by Desbrun and Cani [106]. Their approach adaptively handles highly deformable models through a refining step applied to the particle resolutions. Similarly, Adams et al. [94] proposed an adaptive sampling method focusing on the computational efforts of complex geometric regions of a fluid volume. They reduce the number of particles according to their visual contributions. Adaptive sampling on fluid simulations has also been used to improve and preserve the visual appearance of thin sheets of fluids. After identifying the thin portions of the fluid, the approach introduced by Ando et al. [95] resamples these critical regions based on the anisotropy of the particle neighborhoods. On a different note, Orthmann and Kolb [128] used a temporal blending technique to reduce the number of particles within low-resolution regions. Horvath and Solenthaler [108] improved the conservation of mass for resolution refining methods of particle-based fluid simulation. Meanwhile, Zhu et al. [151] proposed a new grid structure to extend the domain dimensions while preserving the fine details of a fluid simulation. More recently, Aanjaneya et al. [93] also introduced an efficient and sparse data structure to leverage the adaptivity of large-scale simulation domains. Recently, Sato et al. [132] extended the narrow band FLIP method to adaptively allow a transition between the particle band and the grid-based simulation anywhere in the domain (i.e., not exclusively close to the surface). Lastly, inspired by leveraging the data structure adaptively toward multi-scale fluid simulations, Ibayashi et al. [109] introduced a novel warping grid in order to dynamically deform a regular grid evolving during the simulation. While using spatial multi-scale resolution with particles improves visible surface details of liquids, their adaptive nature makes it difficult to exploit a trade-off between procedural and physics-based methods as our approach offers.

Although the resolution dependency is well known in the field of fluid simulation for computer graphics, some challenges remain. Re-computing the forces and the advection step of a small portion of the liquid within a given simulation could be significantly expensive and particularly demanding to preserve its stability. Some authors have shown that this can be improved by using a machine learning process [118, 104, 144, 147]. Compared to these data-driven methods, our approach only requires a coarse liquid simulation to generate high-resolution results. Closer to our motivation, procedural methods have proven that upscaling a simulation can produce interesting small-scale features [116, 122]. In contrast to these procedural methods, our approach is able to locally generate splash details that were poorly approximated due to a coarse discretization of the input simulation domain.

3.2.2. Diffuse Material and Splash Modeling

Several researchers have focused their work on handling interactions of air-liquid mixtures. The results of these interactions generate phenomena such as foam, spray, and white water at the interface of the liquid. Several papers have proposed hybrid schemes to handle sub-scale details with particles advected through a grid-based simulation [140, 119, 123, 145]. A purely Lagrangian approach was proposed by Ihmsen et al. [110]; they used post-processed layers of diffuse particles on top of an SPH simulation to represent foam and spray effects.

In our method, we handle the diffuse portions of a liquid as ballistic particles and refine their behavior with localized artificial pressure corrections from seeded tracers among the upsampled particles.

The diffuse air-liquid interactions are more likely to happen within splashes and highly turbulent regions of liquids. Kim et al. [115] focused their efforts on identifying under-resolved regions and adding massless markers in splashing portions of the liquid. Chentanez and Müller [103] also proposed a solution to improve the visual appearance of liquids for real-time application. Their hybrid model uses spray particles to approximate the liquid-air interactions between rigid and soft bodies. Recently, Um et al. [144] introduced a novel data-driven method to increase the realism of diffuse effects on splashing liquids. A perceptual study was also published on these splashing behaviors [143]. However, in contrast with our work, we are focusing solely on improving the splash modeling before its diffuse effects. Our approach focuses on key events prior to splashing behaviors.

3.2.3. Wave Simulation

Many researchers have considered using a wave simulation instead of a computational fluid to simulate liquid motion [114, 150, 141, 105]. Through the years, they have presented several variations to reproduce waves with the capillary wave equation [114], the *iWave* model [141], and the shallow-water equation [146]. These methods were used to represent a very large body of liquid (e.g., an ocean) as well as to enrich fine-scale details in liquid simulations.

Particle-based waves were also used to increase high-frequency details of liquid simulations [105, 122, 148]. In these methods, the wave equation is used and seeded through advected particles. However, some of these approaches have experienced difficulties when handling complex scenarios such as splashing and highly turbulent behaviors [150, 105]. Mercier et al. [122] managed to robustly couple a dense point set wave representation to a smooth and meshless surface. Later, Yang et al. [148] proposed a novel implicit linear wave model to enrich SPH simulations. Recently, the dispersion kernels used with wave simulations were improved for real-time applications [102, 113]. Similarly, we handle interactions between the upsampled particles and the input coarse liquid with a point-based wave representation and update the input particle velocities and positions.

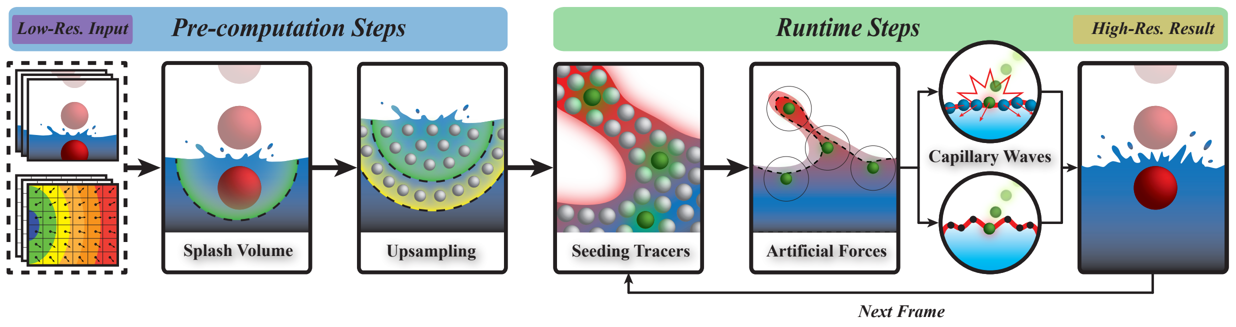


Figure 3.2. Overview of our pipeline. The particles and the velocity field of an existing animated coarse liquid are used as initial inputs. The first two steps are part of a pre-computation process to determine and upsample a splash volume. The simulation steps iterate on the tasks to seed tracers among the upsampled particles, apply a localized artificial pressure, and generate linear waves at impact particle locations on the coarse liquid surface. The highly detailed splashes generated by our approach are computed at a much lower cost than would have been necessary to simulate similar splashes at high resolution with traditional approaches.

3.3. Method Overview

We briefly describe in this section each step of our pipeline (Fig. 3.2). As we aim to benefit from hybrid methods, we decided to rely on the widely known FLIP-based fluid that we use as a coarse input to our approach.

From a coarse simulation as input data (e.g., particles and a velocity field for each frame of the input simulation), two pre-computation steps are performed. First off, we compute a splash volume (§ 3.4.1) by combining the intersection between user-defined colliding volumes and the input volume (i.e., generated from the coarse input liquid). As presented in Fig. 3.2, the splash volume (e.g., green portion of the upsampling step) is slightly extended (e.g., yellow portion of the upsampling step) to fully cover disturbances surrounding solid-liquid and liquid-liquid interactions. Thereafter, the resulting volume is discretized and upsampled using a higher density of particles per cell (§ 3.4.2). These upsampled particles are then injected into the existing velocity field and advected alongside the input particles of the coarse liquid (§ 3.5.3). Among upsampled particles, so-called tracers are seeded (§ 3.5.1) and used as control particles to apply localized artificial forces mimicking the effect of surface tension (§ 3.5.2). Lastly, we use a point-based linear wave model to couple interactions between the upsampled particles and coarse particles at impact locations. The waves are seeded to enrich these impacts over the coarse input liquid (§ 3.6). As shown in Fig. 3.2, two pre-computation steps are performed only once for the whole input simulation data and can be reused with different parameter sets to generate alternative high-resolution results.

3.4. Splash Volume

A subset of the input simulation domain is determined to locally increase details around potentially splashing portions of the liquid. This volume is computed from the coarse input simulation data. At each frame of the input simulation, we identify a colliding volume that will contribute to the *splash volume* (§ 3.4.1). Thereafter, the splash volume is refined to upsample important portions of the liquid with more particles (§ 3.4.2).

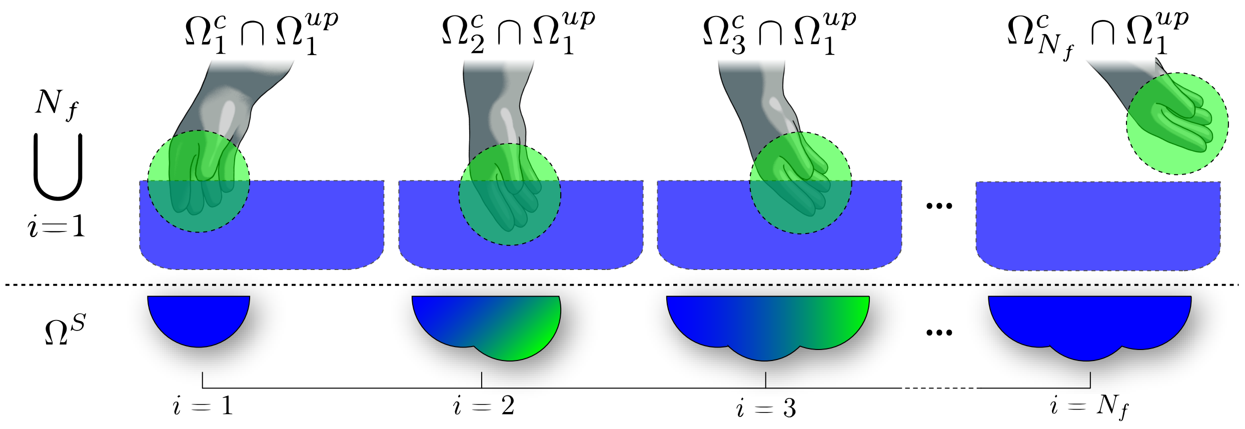
3.4.1. Splash Volume Generation

From the input liquid, our method splits the scene into *upsampling* and *colliding* volumes. These distinct volumes are determined manually at the first frame of the input animation. The colliding volumes can be either a volume of liquid or a static or dynamic obstacle. We identify the upsampling volume as Ω^{up} and the colliding volumes as Ω^{c}.

The splash volume Ω^{S} is obtained from the unions of the upsampling volume Ω^{up} and the colliding volumes Ω^{c} over time. The splash volume Ω^{S} is therefore pre-computed from all N_f frames of the input simulation and is expressed as

$$\Omega^{S} = \bigcup_{i=1}^{N_f} \left(\Omega_i^{c} \cap \Omega_1^{up}\right), \tag{3.1}$$

which corresponds to the intersection between the upsampling volume Ω^{up} and the union of the colliding volumes. The updating process of the splash volume Ω^{S} is illustrated in Fig. 3.3 as well as in the accompanying video. As shown in the top row, we only keep the intersection with the volume Ω^{up} at the first frame $i = 1$.



$\Omega_1^c \cap \Omega_1^{up}$
$\Omega_2^c \cap \Omega_1^{up}$
$\Omega_3^c \cap \Omega_1^{up}$
$\Omega_{N_f}^c \cap \Omega_1^{up}$
$\bigcup_{i=1}^{N_f}$
...
Ω^S
...
$i=1$
$i=2$
$i=3$
$i=N_f$

Figure 3.3. Generation of the splash volume from the upsampling volume (blue) and the collision volumes (green).

Our method for generating a splash volume handles scenarios in which solid-liquid or liquid-liquid collisions occur. For solid-liquid collisions, we convert the input mesh of the static or dynamic obstacle into a volume Ω^c. As shown in Fig. 3.3, only the intersection with Ω^{up} is kept since it will ultimately represent the splash volume Ω^S to be upsampled. For liquid-liquid interactions, we split the volume of the input liquid into distinct volumes: one volume Ω^{up} to be upsampled (e.g., liquid container), and volumes Ω^c that will collide with the upsampled volume (e.g., a falling drop of liquid). For this type of interaction, we keep track of the particles initially composing Ω^c to generate the deforming volume Ω_i^c over time.

3.4.2. Particle Upsampling

The purpose of the oversampling step is to identify regions within the splash volume Ω^S in which upsampled particles should be added (see § 3.7 for implementation details). As opposed to upsampling entirely the volume Ω^S, our upsampling method focuses on regions (i.e., subset of cells) in which significant changes are likely to occur (see § 3.8.6 for a comparative analysis). Our *emerge-tendency* upsampling method concentrates on the regions of the splash volume in which particles are most likely to emerge. We define the emerge-tendency of cells as their probability to contain particles that reach the surface at some point during the entire animation. A likelihood score is computed for each cell identifying which ones to upsample. The likelihood score is obtained by computing three reward terms as follows:

$$\Theta(x) = \sum_{i=1}^{N_f} \mathrm{rt}_i(x) + \alpha \sum_{i=1}^{N_f} \mathrm{ru}_i(x) + (1-\alpha) \sum_{i=3}^{N_f} \mathrm{rs}_i(x), \tag{3.2}$$

where Θ is a set of cells containing the score associated with each grid cell x within Ω^S and in which the reward terms are respectively based on particle trails rt, velocity norm $\|\vec{u}_i(x)\|^2$ noted as ru, and velocity orientation stability rs. We also introduce a weighting factor α to the reward terms ru and rs controlling the relative influence of each other on the likelihood score. We used $\alpha = 0.5$ in all our examples except with the *River* (see § 3.8.2). Given as

Algorithm 4: Identify cells to upsample with Ω^S.

```
for $x \in \Omega^S$ do
    $\mathrm{rt}(x) = \mathrm{ru}(x) = \mathrm{rs}(x) = 0$
    for $i \in [1, N_f]$ do
        if crossSurface($p(x)$, $i$) then
            $\mathrm{rt}(x) \mathrel{+}= \mathrm{rt}_i(x)$
        $ru(x) \mathrel{+}= \|\vec{u}_i(x)\|^2$
    for $i \in [3, N_f]$ do
        $\mathrm{rs}_i(x) = 1$ for $j \in [i-2, i]$ do
            $\mathrm{rs}_i(x) \mathrel{*}= \max(\vec{u}_i^j(x) \cdot \vec{n}_i(x), 0) \geq \cos\frac{\psi}{2}$
        $\mathrm{rs}(x) \mathrel{+}= \mathrm{rs}_i(x)$
    $\Theta(x) = \mathrm{rt}(x) + \alpha\mathrm{ru}(x) + (1-\alpha)\mathrm{rs}(x)$
meanScore = computeMeanScore($\Theta$)
$\mathbf{s} = \emptyset$
for $x \in \Omega^S$ do
    if $\Theta(x) \geq meanScore$ then
        $\mathbf{s} \leftarrow \mathbf{s} \cup x$
UpsampleCells($\mathbf{s}$, $s_{up}$)
```

input the entire animation of the coarse liquid, computing the particle trail reward term rt is fairly trivial. We set $\mathrm{rt}_i(x)$ to be equal to the number of particles $p(x)$ originating from cell x that cross a surface cell within the splash volume at frame i. The surface cells are defined as the discretization of the surface level set. Since our objective is to predict which particles among the upsampled particles will reach the surface, only computing the trails of the coarse particles will be insufficient. In that sense, the input velocity field $\vec{u}$ complements the particle trails. We set $\mathrm{ru}_i(x)$ to the L^2 norm of the grid cell velocity. Consequently, the velocity L^2 norm term penalizes cells containing a lower velocity over time.

Lastly, we complement the likelihood score Θ of each cell x with the reward rs, which considers the grid cell velocity orientation over time. The rs term rewards grid cells with

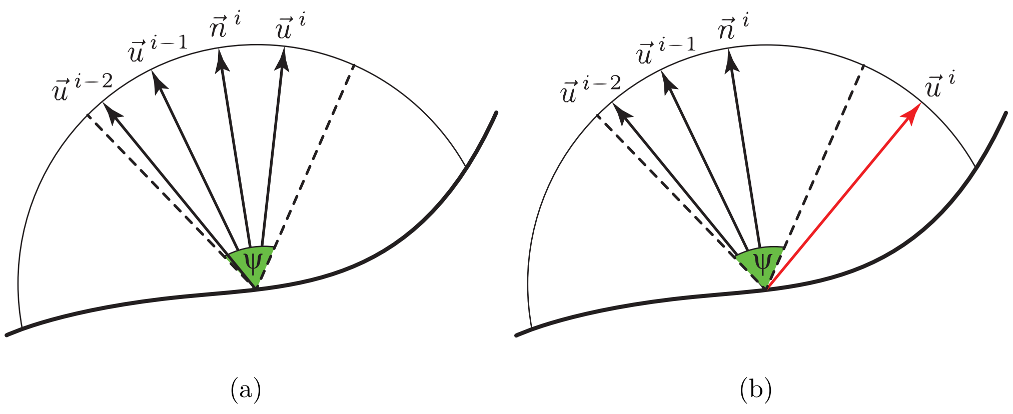


Figure 3.4. 2D example of rs_i evaluation.

velocity orientations pointing toward the liquid interface for a sequence of consecutive frames. The velocity orientation stability $\mathrm{rs}_i(x)$ is equal to 1 if consecutive velocities lie within a cone defined by an angle ψ aligned with the normal of the liquid interface (otherwise $\mathrm{rs}_i(x) = 0$). We figured out (after a few experiments) that a window of three consecutive frames (i.e., $i-2$, $i-1$, and i) was a good compromise to estimate our stability term over time. Consider the 2D example shown in Fig. 3.4: the current and two previous velocities are oriented inside the cone defined by ψ in green (top image: $\mathrm{rs}_i(x) = 1$). The bottom image shows a case where one of the velocities $\vec{u}^{\,t}$ (red) is outside the cone (i.e., $\mathrm{rs}_i(x) = 0$).

The resulting subset of cells **s** to upsample contains the cells with a likelihood score (as shown in Alg. 4) greater or equal to the average likelihood score among the cells contained inside the splash volume Ω^S. Finally, the cells identified are upsampled using the commonly known jittered grid technique. As proposed by Zhu and Bridson [152], we create randomly jittered particles in every cell using d^3 subgrid positions where d is the subgrid dimensions. The number of particles per cell used in our examples corresponds to the number of particles per cell given from the coarse input liquid multiplied by a factor s_{up}. Therefore, the number of particles upsampled is equal to $s_{\mathrm{up}}d^3$.

3.5. Tracers and Upsampled Particles

Tracers are seeded in the splash volume, but only within the narrow band (§ 3.5.1) defined from the interface of the liquid. These tracers are used to influence the upsampled portions of the liquid; they mimic the effect of surface-tension forces observed in splashes. We call them *tracers* because they trace trajectories for upsampled particles. The movement of a tracer is initialized with the velocity transferred from the closest upsampled particle, and afterward, it is only influenced by gravity. The upsampled particles within a user-defined maximum distance of tracers are affected by an artificial pressure term as used with position-based fluids (PBF) [120]. The details of upsampled particles velocity update are presented in § 3.5.3.

3.5.1. Seeding Tracers

Since our aim is to enhance surface details around localized portions of the input liquid, we constrain the seeding of tracers within a narrow band according to a set of criteria. The seeding operation is performed in a way to achieve a uniform distribution, to distribute the tracers close to the interface, and to seed them according to the velocity norm throughout the animation. Tracers are seeded using the Fast Poisson Disk sampling method [100] within a narrow band of the splashing liquid. The narrow band is defined as a set of cells discretized from the liquid interface. We use the 3D implementation of the sampling method considering the narrow band thickness (see § 3.7 for implementation details). The band thickness $\triangle\tau_{\text{tp}}$ is defined as twice the influence radius of the tracers (refer to Table 3.1). This band thickness was chosen to ensure a minimum number of upsampled particles affected by our tracers and to avoid neighborhood deficiency.

The last criterion on the velocity norm is used to discard insignificant motionless portions of liquid. The velocity norm criterion focuses on seeding tracers in regions where the input liquid is likely to be disturbed and to present notable small-scale detail differences. Tracers are seeded at locations where the L^2 norm of the velocity is increasing. In other words, tracers are seeded in cells with increasing velocities over time. The moment of creation of tracers is determined by the velocity criterion as well. To trigger the seeding process, we use a five-frame temporal sliding window $\triangle w_{\text{tp}}$ and evaluate the velocity L^2 norm change for every grid cell. The L^2 norm evaluation of the velocities is performed as shown in Alg. 5.

Although evaluating velocities within a five-frame window leads to very good results on our examples, this frame interval could be adjusted by the user. Cells where tracers will be seeded must also have a high velocity; the velocity L^2 norm must be higher or equal to the current average velocity $\overline{\|\vec{u}_i\|}$ in order to exclude small velocities increasing over time. We

Algorithm 5: Evaluate if the velocity L^2 norm of a cell at time t satisfies the tracer seeding conditions within the temporal frame interval $[t, t + \triangle w_{\mathrm{tp}}[$.

```
for $i \in [t, t + \triangle w_{tp}[$ do
    if $\|\vec{u}_{i+1}\|^2 \leq \|\vec{u}_i\|^2$ or $\|\vec{u}_i\|^2 < \overline{\|\vec{u}_i\|}^2$ then
        return false
return true
```

update an average velocity from all the cells containing particles at each frame and use it as a reference. Seeding tracers is computed at a negligible cost since it is performed exclusively on cells at the interface and inside the splash volume.

3.5.2. Localized Artificial Forces

We use the tracers to apply a correction term on the position of the upsampled particles to avoid artifacts such as clustering and clumping. These artifacts issued from negative pressures occur at the initialization step with the input velocity field. We use an approach similar to that of Monaghan [124] to correct tensile instability with an artificial pressure term, as follows:

$$s_p = -k \left(\frac{W(\vec{p}_i - \vec{p}_j, h)}{W(\vec{p}_i - \vec{q}, h)} \right)^n, \tag{3.3}$$

where $\vec{q}$ is a point inside the smoothing domain defined between $\vec{p}_i$ and $\vec{p}_j$, n the pressure power constant, and k a stiffness constant (we always use $k = 0.1$ and $n = 4$). The particle displacement $\triangle \vec{p}_i$ based on the artificial pressure term s_p is then expressed as follows in the particle position update step, as used by Macklin and Müller [120]:

$$\triangle \vec{p}_i = \frac{1}{\rho_0} \sum_j s_p \nabla W(\vec{p}_i - \vec{p}_j, h). \tag{3.4}$$

The support radius h_{tp} is defined between 0 and the input support radius h to adjust the influence of tracers over the upsampled particles. The value of h_{tp} depends on the desired visual outcome (as shown in Fig. 3.16). Furthermore, the particle density is usually unknown when using the FLIP simulation method. We thus derived a normalized expression implicitly taking density into account [138]:

$$\triangle \vec{p}_i = \frac{1}{\sum_j W(\vec{p}_i - \vec{p}_j, h)} \sum_j s_p \nabla W^*_{\mathrm{tracer}}(\vec{r}_{ij}, h_{\mathrm{tp}}). \tag{3.5}$$

Similarly to Müller et al. [125], we use the *Poly6* kernel for density estimation and the *Spiky* kernel for gradient estimation. In Eq. 3.5, we employ a slightly different kernel function to compute the position update. We propose a tracer-relative kernel function as follows:

$$W^*_{\text{tracer}}(\vec{r}_{ij}, \vec{r}_{it}, h_{\text{tp}}) = \begin{cases} (h_{\text{tp}} - \min(|\vec{r}_{ij}|, |\vec{r}_{it}|))^3 & 0 \leq |\vec{r}_{it}| \leq h_{\text{tp}} \\ 0 & \text{otherwise,} \end{cases} \tag{3.6}$$

where $|\vec{r}_{ij}|$ is the length of the vector between upsampled particles i and j, and $|\vec{r}_{it}|$ is the length of the vector between upsampled particle i and the closest tracer. By promoting the closest tracer, we ensure to improve a localized particle attraction based on the most significant contributor. In the proposed approach, tracers are used as control particles over the upsampled particles. Their kernel-based contribution influences the neighboring upsampled particle positions by updating their velocities and by applying an artificial force to mimic the pressure nature of liquids on diffuse particles.

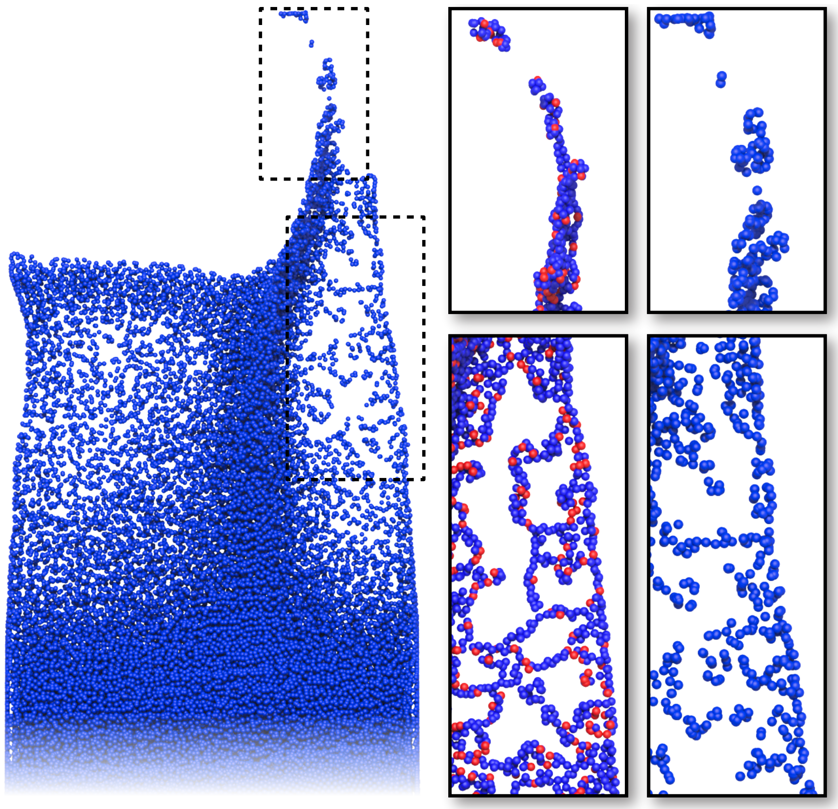


Figure 3.5. Our tracers (red) constrain an artificial pressure term integrated with PBF. The localized correction produces a smooth and controllable surface-tension effect among the upsampled particles (left) compared to the originally diffuse behavior (right).

By comparison to position-based fluids (PBF) (Eq. 3.3), we use a point $\vec{q}$ defined as the position of the closest tracer. In contrast with the one used by Macklin and Müller [120], our artificial pressure term is primarily applied to particles within a radius distance from the seeded tracers. The distance between an upsampled particle and the closest tracer must be smaller or equal than the support radius h used for the coarse input. This term improves the generation of fine surface-tension details among the upsampled particles.

Our tracer-relative kernel function can also be easily integrated into the PBF model. As shown in Fig. 3.5, seeding tracers among the particles and applying our constrained artificial pressure can locally improve affected regions in a PBF liquid.

3.5.3. Upsampled Particles: Advection and Lifespan

During their lifespan, upsampled particles evolve between different states, based on the types of details they are likely to add to the simulation. In this section, we will explain the three states (*bulk*, *ballistic*, and *impact*) and how particles evolve between states (Fig. 3.6).

The bulk state is flagged when an upsampled particle is inside the splash volume. By default, all upsampled particles are considered as bulk at the initialization step (i.e., when seeded at pre-computation steps). The bulk particles p_{bulk} are advected using a Runge-Kutta scheme through the input velocity field [152]. The velocity update used for bulk particles is performed solely using the input coarse velocity field as follows:

$$\vec{u}^{\,i}_{p_{bulk}} = \vec{u}^{\,i-1}_{p_{bulk}} + \mathrm{lerp}(\triangle\vec{u}^{\,i}_{input}, \vec{x}^{\,i}_{p_{bulk}}). \tag{3.7}$$

As with FLIP, the weighted average of the velocities of the upsampled particles is computed and added to the nearby staggered MAC grid $\vec{u}^{\,i}_{input}$ nodes. At each step, the interpolated difference from the input grid velocity $\triangle\vec{u}^{\,i}_{input}$ is computed and assigned to each particle at their respective position $\vec{x}^{\,i}_{p_{bulk}}$. This velocity update step is very effective since we are computing these velocity changes only for cells inside a subset (i.e., splash volume) of the input simulation domain.

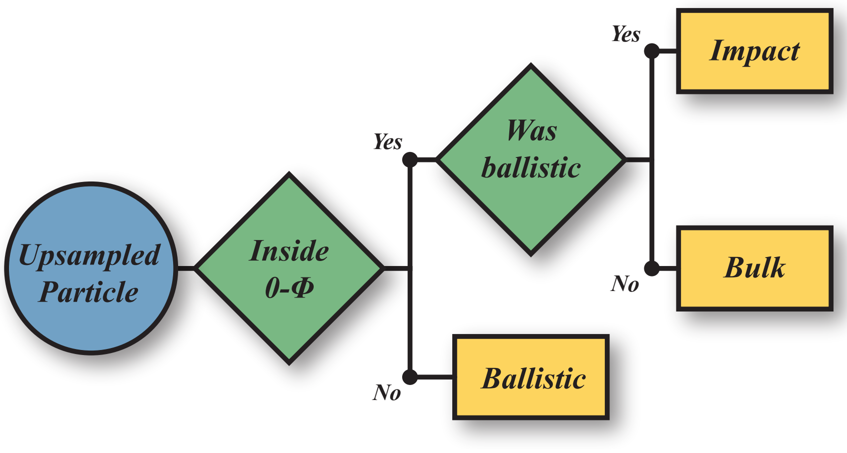


Figure 3.6. State machine (trigger events in green) for determining the particle state (yellow).

The ballistic state is triggered when an upsampled particle leaves the bulk volume determined by the liquid interface of the input simulation (see Fig. 3.6). The upsampled particles flagged as ballistic are initialized with the current particle velocity $\vec{u}^{\,i}_{p_{bulk}}$ previously computed before leaving the coarse liquid (i.e., w.r.t. the liquid surface). They are later influenced by gravity and possibly by tracers (as previously described in § 3.5.2) as follows:

$$\vec{u}^{\,i}_{p_{bal}} = \vec{u}^{\,i-1}_{p_{bal}} + \vec{g}\triangle t, \tag{3.8}$$

and where the updated position of a bulk particle $\vec{x}^{\,i}_{p_{bal}}$ at frame i is obtained by integrating the resulting $\vec{u}^{\,i}_{p_{bal}}$ with the correction term defined in Eq. 3.5.

Lastly, the impact state is triggered when a ballistic particle intersects back with the coarse liquid surface. The name *impact* is taken from the fact that these particles will alter the velocity of the particles within the bulk volume. This state is only flagged for a particle during one frame. It allows us to detect the positions for seeding point-based waves within the bulk portion of the input liquid (§ 3.6). As soon as we store the impact position, the particle is flagged back as bulk and its velocity is updated according to the underlying velocity field. We use trilinear interpolation for a smooth transition of the velocity from its ballistic nature to the current grid cell.

For efficiency reasons, we discard particles that are unlikely to contribute to splashes. We define an upsampled particle as discarded if it no longer contributes to the surface details of the liquid. We use two criteria to determine if an upsampled particle contributes to surface

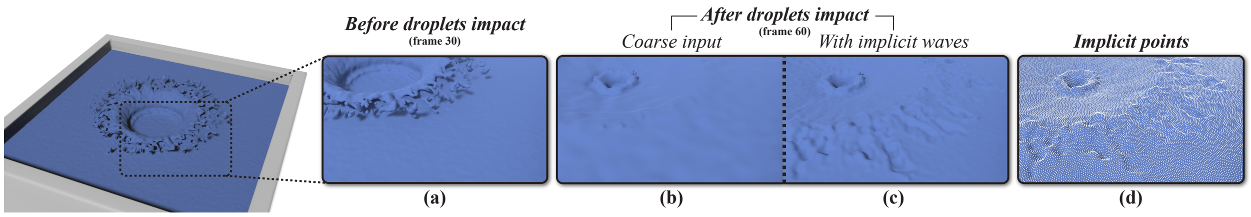


Figure 3.7. The small droplets of a splash (a) are often absorbed on impact (b), resulting in loss of multiple fine surface details. Our implicit linear wave model provides convincing interactions through small swirls generated by these droplets (c). The waves are propagated through the implicit points projected on the surface (d).

details: its distance to the interface inside the liquid and its current velocity. With these two criteria, we manage to preserve small- and large-scale details around induced splashes. The distance criterion uses the same distance as defined when seeding tracer particles (§ 3.5.1). However, this threshold does not apply if the upsampled particle is still inside the splash volume. As a second criterion, we also discard upsampled particles if their velocity is lower than the average velocity. We use the same average velocity per cell obtained when seeding tracers (see § 3.5.1).

3.6. Implicit Linear Waves

Since ballistic particles are likely to touch the interface of the coarse liquid, additional details must be generated on the liquid interface in order to preserve the realism of the approach. Following the detection of impact particles (i.e., ballistic particles intersecting the coarse liquid surface), we handle their interactions with the coarse liquid provided as input (Fig. 3.8). In our approach, these interactions are reflected as velocity updates on the grid cells, and generated as seeded waves evolving from the impact locations. These waves are modeled by surface points uniformly distributed to cover the implicit surface. A wave displacement is computed according to the impact properties and the estimated normals obtained from the implicit surface. The evolving waves locally affect the neighboring cell velocities by their tangential velocities. The neighboring affected area is determined by the impact radius. The positions of the coarse input particles are then updated to reflect these changes (see Fig. 3.7).

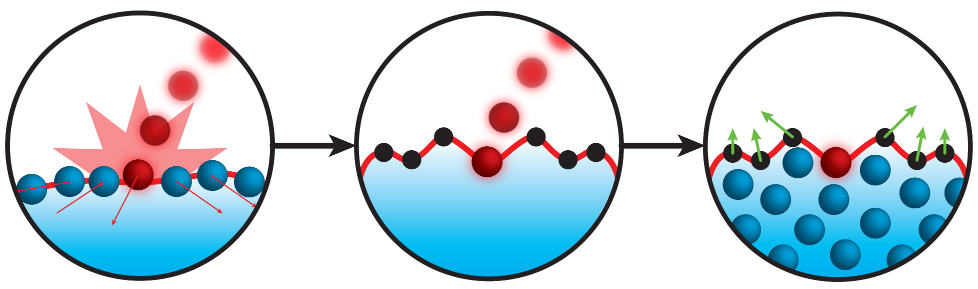


Figure 3.8. An impact is defined by the intersection between an impact particle and the coarse liquid (left). Surface points distributed and projected on the surface are used to support and evolve the linear waves (middle). The tangential velocities of the point-based waves are used to update the underlying grid velocity, and then the positions of the input particles from the coarse liquid (right).

3.6.1. Surface Points

The surface points are distributed uniformly over the entire implicit surface generated from the coarse liquid. These surface points are generated once at the very first frame of the input animated fluid. We use a Poisson distribution to initialize the surface points along the surface. The positions of the surface particles from the coarse input liquid are used as the initial locations for Poisson disk sampling [100]. The surface points that we call *implicit points* from this stage, are then projected on the surface of the coarse liquid. The number of implicit points created is bounded by the particle density per cell of the input liquid. Since we aim to solely introduce small impact interactions, our wave model does not require a computationally expensive maintenance process (as opposed to Mercier et al. [122]) on implicit points after the projection step. In addition, it is noted that surface points are not used when generating the surface mesh (as shown in Fig. 3.8(c)) considering that their sole purpose is to perturb surrounding cells of the input velocity grid.

3.6.2. Wave Seeding and Propagation

The seeding locations are determined by the position of the upsampled particles flagged as impact particles. We use the current velocity magnitudes of the upsampled particles at impact locations to parameterize the wave properties when seeded. Since our goal is to infer interactions from these impacts, our wave model is defined as an efficient gridless version of the 2D heightfield liquid simulation [126]. With this model, height is the factor that determines the wave velocity. On impact, we update the heights $h_{p_{\text{impl}}}$ of the implicit points (p_{impl}) i within radius as follows:

$$h^i_{p_{\text{impl}}} = h^i_{p_{\text{impl}}} + \frac{1}{N} \sum_{j=0}^{N-1} \left[\cos\left(\frac{\|\vec{c} - \vec{p}^j\|}{r_{\text{im}}} \pi \right) \alpha_s \|\vec{u}_{\text{im}}\| \right], \tag{3.9}$$

where $\vec{c}$ is the center of the impact position, $\vec{p}^j$ the position of the j^{th} implicit neighbor of N points, and $\vec{u}_{\text{im}}$ the velocity of the impact weighted by a user-defined normalized scaling factor α_s. The radius of the impact r_{im} is also defined as a function of the impact velocity norm $\|\vec{u}_{\text{im}}\|$ as follows

$$r_{\text{im}} = \alpha_r \|\vec{u}_{\text{im}}\|, \tag{3.10}$$

and weighted by a user-defined normalized constant α_r. In our examples, we used an α_r equal to 10% of the simulation scale from the coarse input liquid. The first term in brackets of Eq. 3.9 is the height displacement applied by an implicit neighbor point j, and the second term is basically the strength of the wave displacement.

We propagate and evolve the waves using the neighbor contributions at each frame. For each implicit point i, an average height $\bar{h}_{p_{\text{impl}}}$ from the neighborhood is computed and used to update a local 1D wave velocity (initialized as zero before impact):

$$u^i_{p_{\text{impl}}} = u^i_{p_{\text{impl}}} + \frac{1}{N} \sum_{j=0}^{N-1} \Big[(1 - \beta_{\text{damp}}) u^j_{p_{\text{impl}}} + 2(\bar{h}_{p_{\text{impl}}} - h^j_{p_{\text{impl}}}) \triangle t \Big]. \tag{3.11}$$

The updated height is obtained by adding this 1D velocity to its current value $u^i_{p_{\text{impl}}}$ (initialized by the grid velocity at that position). The term $(1 - \beta_{\text{damp}})$ enforces wave damping. Finally, we update the positions on every implicit point affected, and according to the orientation of the implicit point normal . The normal at these points is computed using an averaged least-squares planar fit of the local gradient of Φ based on the implicit point neighbors. The planar fit allows us to compute a plane defined by a tangent and a bi-tangent, both orthonormal to its normal.

3.6.3. Influence over Coarse Particles

We achieve the update on the input surface particle positions by converting the affected particles to bulk particles. The subset of affected particles is determined by the impact radius. As a wave is affected by damping, the corresponding particles return progressively to their current input state. The implicit point velocities $u_{p_{\text{impl}}}$ are used to alter the coarse grid velocity, and to update the affected coarse particles. We use the tangential velocity at each implicit point defining the waves. The tangential velocity is computed by orienting the implicit point velocity towards its tangent vector. For our wave model, we chose the tangent vector $\vec{t}^{\,i}_{p_{\text{impl}}}$ according to the impact velocity orientation and with respect to $\vec{n}^{\,i}_{p_{\text{impl}}}$. We compute the updated grid cell velocity by interpolating it between its current value and the tangential velocity at that position (i.e., $\vec{x}_{p^i_{\text{impl}}}$). The grid cell velocity is returning to its unmodified value as the tangential velocity gets attenuated as:

$$\vec{u}^{\,*}_{\text{input}}(x) = (1 - \gamma_{\text{att}})\vec{u}_{\text{input}}(x) + \gamma_{\text{att}}(u^{\,i}_{p_{\text{impl}}}\vec{t}^{\,i}_{p_{\text{impl}}}). \tag{3.12}$$

The normalized interpolation term γ_{att} is calculated based on the ratio between the current implicit point velocity at frame t and the one when the wave has been seeded ($\vec{u}^{0}_{p_{\text{impl}}}$):

$$\gamma_{\text{att}} = \frac{\|\vec{u}_{p_{\text{impl}}}\|}{\|\vec{u}^{0}_{p_{\text{impl}}}\|}. \tag{3.13}$$

The velocity updates are performed exclusively on the surface cells (i.e., according to the discretized level set).

3.7. Implementation

Most of the steps of our approach are computed on the GPU with CUDA. Our examples were generated on an Intel i7 quad core running at 3.4 GHz with 16 GB of RAM, and using an NVIDIA GTX 1080 with 8 GB GDDR5X. Our input coarse simulations come from the implementation of FLIP available in Houdini 16.5 [136]. Although our input is obtained from a traditional FLIP method, the use of any of its variants, such as narrow-band FLIP [107], would have been equally valid. The parameter values used with the presented examples are exposed in Table 3.1. In the following, we also include implementation details for core steps of our method to facilitate reproducibility of the presented results.

Parameter	Notation	Value
Input support radius	h	$*$
Scaling reward terms	α	$[0.5, 0.7]$
Subgrid dimensions	d	$[2, s_{up}d]]$
Tracer influence radius	h_{tp}	$[0, h]$
Tracer band thickness	$\triangle\tau_{\text{tp}}$	$2h$
Tracer temporal window	$\triangle w_{\text{tp}}$	5
Tracer pressure power	n	4
Tracer stiffness constant	k	0.1
Upsample factor	s_{up}	$[2, 6]$
Scaling impact velocity	α_s	0.5
Scaling impact radius	α_{r}	$0.1s_{\text{up}}$
Scaling wave damping	β_{damp}	$[0.7, 1.0[$

Table 3.1. Parameters used to generate our results. As exposed, our approach has only few user-defined parameters. Most of the parameters presented are fixed (i.e., invariant from the types of scenario) or given from the input liquid. * The input support radius h and scaling upsample factor s_{up} are the only parameters that are specific to a scenario; they depend on the scale and the level of detail required by it.

Splash Volume. When generating the splash volume, Ω^{up} and Ω^c are computed using an SDF of the input particles. During SDF computation, we also consider as boundaries the input obstacles such as static and dynamic meshes. The discretized splash volume resolution corresponds to the resolution of the input velocity grid. It is formed as a subset of cells of the input velocity grid. Of course, we solely consider cells containing particles as part of this subset. In the case of liquid-liquid interactions, we keep track of the distinct volumes of liquid by assigning the particle unique IDs to a volume ID. Naturally, and for that reason, we avoid using a resampling step when generating the input liquid simulation. For both solid-liquid and liquid-liquid interactions, we expand the colliding volume Ω^c in order to capture properly the disturbance on the upsampling volume Ω^{up}. Expanding Ω^c is particularly helpful with solid-liquid interactions since obstacles tend to push particles outside their boundaries.

Moreover, we can automatically trigger to stop updating the splash volume Ω^S if it remains constant after several consecutive frames (e.g., as with the *Hulk's wrath* example shown in Fig. 3.13). We observed that while we should limit the number of cells in the splash volume, the impact on the total computations is less critical than other parts of our method. Lastly, we determine the first frame i from which we track the splash volume as the first collision detected between the colliding volume Ω^c and the upsampling volume Ω^{up}.
Volume Upsampling. In our implementation, we upsample Ω^S by using its discretized representation (i.e., set of cells). The input particles contained in the splash volume are automatically converted to upsampled particles. Therefore, we are able to prevent undesired artifacts that may occur by separately processing upsampled particles from the low-resolution input liquid. Using eight particles per grid cell is usually enough to extract fine details with the FLIP simulation model. As discussed by Zhu and Bridson [152], using more particles will rarely generate better results. However, since our approach aims to upsample an input coarse liquid for visual improvements, using locally more particles can greatly increase the small-scale details of the results. In fact, for the purpose of generating dynamic details for splashing liquids, we show that a higher density of particles will locally produce high-detailed and realistic results.
Seeding Tracers. As described by Bridson [100], we chose k to define the density of tracers for each sample x_i ($k = 5$ was used in all our examples). The samples x_i are initially defined at each center of the cells contained in the discretized narrow band of the liquid surface. Finally, the same distance as used with $\triangle\tau_{\text{tp}}$ is employed for the Poisson disk radius r. For effeciency, a tracer is discarded as soon as it touches the interface of the bulk liquid.

3.8. Results and Discussion

Our post-processing approach can locally generate realistic splashes in a wide range of scenarios. These splashes are obviously different than the ones that would result from a simulation with denser particles everywhere, but they are representative of such simulations since they affect both the splash details and the coarse liquid surface. We can handle induced dynamics for solid-liquid as well as liquid-liquid interactions. In the following, we discuss the results obtained for the different examples presented in this paper and its supplemental material.

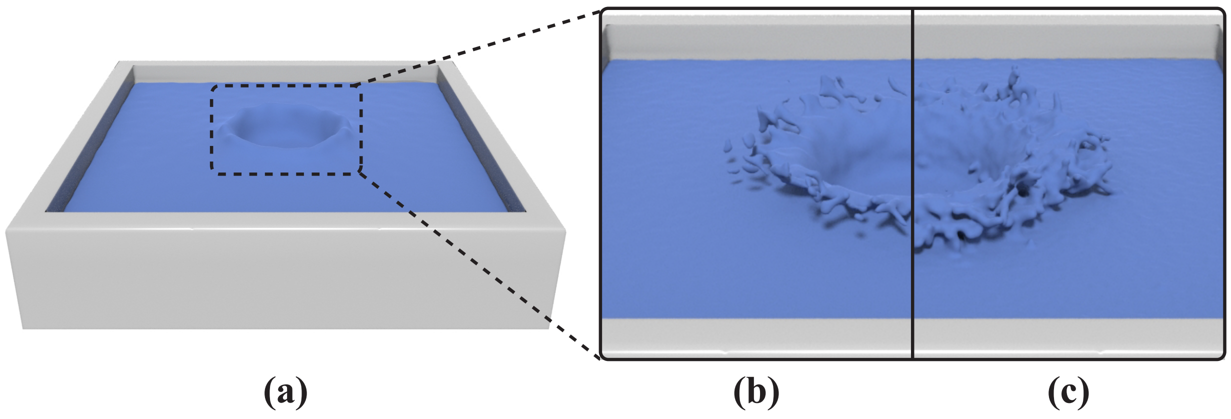


Figure 3.9. *Spherical drop*: (a) Low-resolution (95k particles, 40^3 grid resolution, 0.9 s/frame) simulation of a spherical drop of liquid falling in a pool of liquid. (b) High-resolution (1.2M particles, 200^3 grid resolution, 7 s/frame) simulation. (c) Our method (115k added particles, 0.3 s/frame).

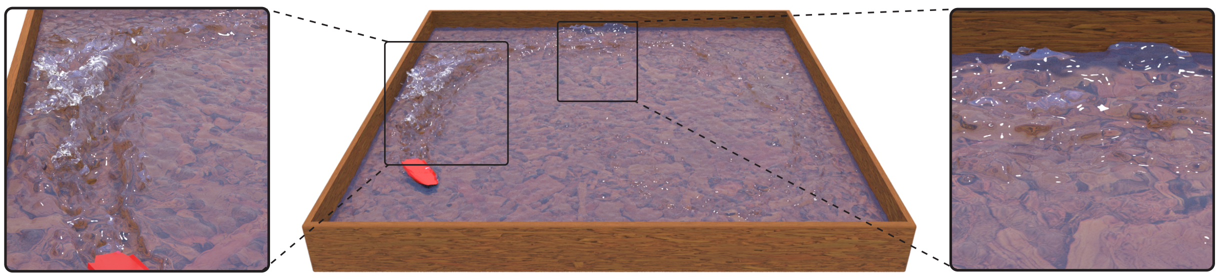


Figure 3.10. Induced splash dynamics is generated on top of a coarse FLIP liquid provided as input. Our approach is able to reproduce localized realistic splashes (left enlarged) that were initially absent in the input coarse liquid. We also handle interactions between our geometric model and the coarse input, such as reflected swirls and waves (right enlarged), by adding an implicit linear wave model.

3.8.1. Comparison with High-Resolution Results

The idea behind our approach is to provide a sufficient density of particles locally to reduce the noticeable numerical dissipation occurring at sudden disturbance locations in the liquid. Our method focuses on generating splashes locally where they should appear. As shown in Fig. 3.9(a), the low-resolution liquid is unable to represent the small crown splash because of its very coarse discretization (95k particles). We had to increase the number of particles to slightly above a million (Fig. 3.9(b)) to successfully generate a fine-detailed splash with a traditional FLIP method.

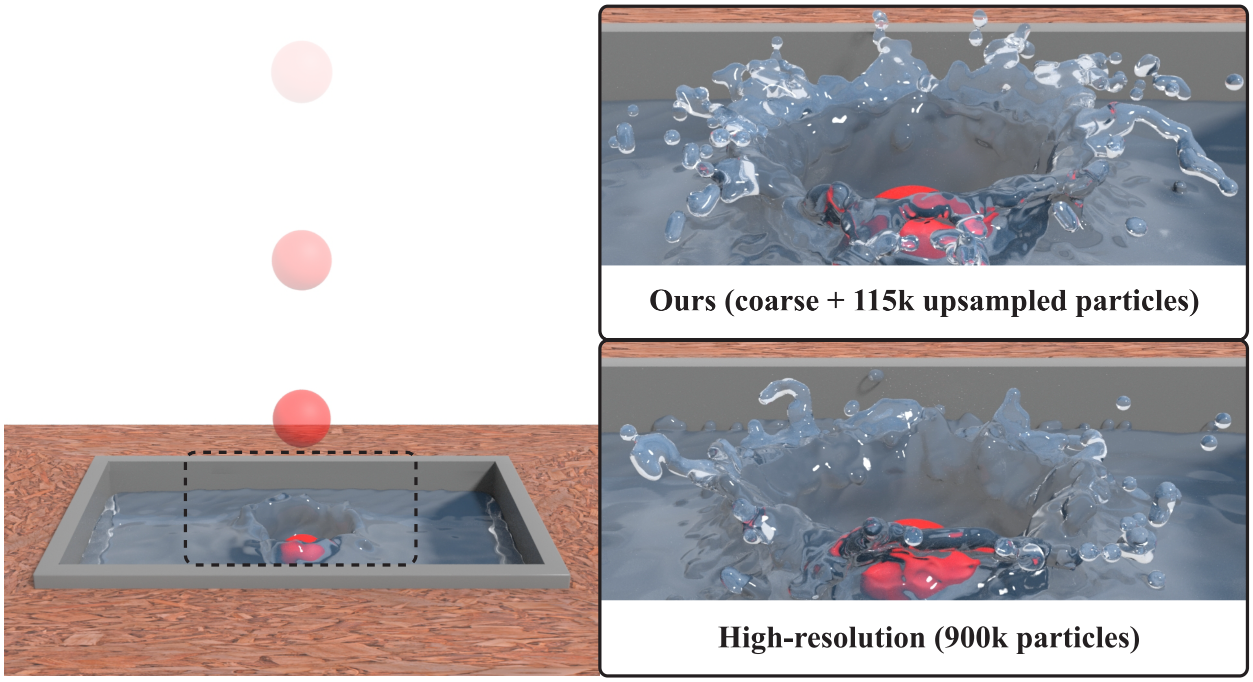
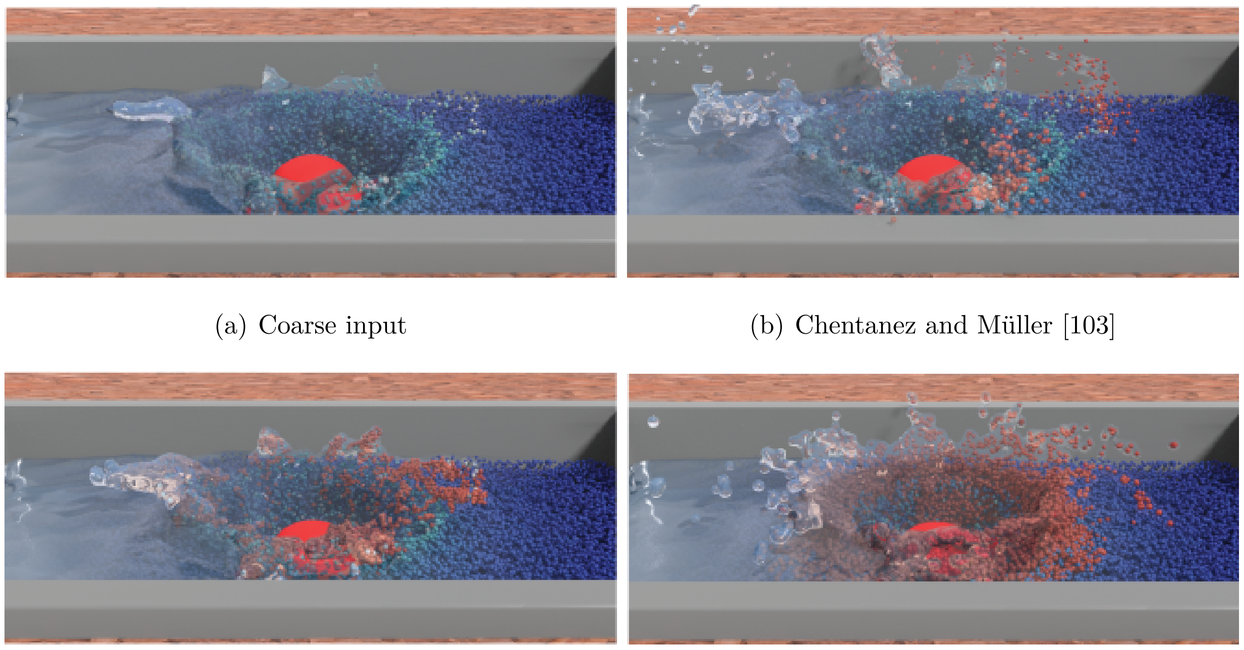

Ours (coarse + 115k upsampled particles)
High-resolution (900k particles)

Figure 3.11. A ball falls in a small container of liquid. As clearly noticeable in the low-resolution example (left), the splash is totally nonexistent. Despite using this same coarse liquid as an input to our method, we managed to generate a detailed crown splash (top right), that is similar to the high-resolution reference (bottom right).

Despite the fact that the standard FLIP method provided enough fine details to extract a decent splash at high resolution, the computational cost of simulating this example was high (∼7 s/frame) compared to the low-resolution version (∼0.9 s/frame). Although this has been solved for hybrid methods, it is still worth mentioning that most of the high-resolution particles barely contributed to the desired result, even within a narrow band (e.g., [107, 131]). Our experiments on this example showed that only around 100k of these particles from the high-resolution simulation actually contributed to the central splash. With our post-processing approach, we need to add only 115k particles (based on criteria provided in § 3.4.2) on top of the coarse input particles, to generate a realistic splash reproducing a behavior matching the one found in the high-resolution simulation. Solely a small portion of the simulation domain needed to be processed by our method at a very inexpensive cost of



(a) Coarse input

(b) Chentanez and Müller [103]

(c) Ihmsen et al. [110]

(d) Ours

Figure 3.12. Comparison between our approach and state-of-the-art methods using spray particles. The spray particles ((b) and (c)) and our upsampled particles (d) are identified in red.

about a second per frame from the coarse input liquid. Clearly, an arbitrary larger simulation domain would have resulted in larger gains.

3.8.2. Solid-Liquid Interactions

Our method has a fully automatic way to deal with solid-liquid interactions. The animated mesh of the solid interacting with the liquid is used to generate the associated splash volume. This step is computationally inexpensive and the frames only need to be processed once to compute and upsample the resulting splash volume. Another advantage of our approach is that once the pre-computation steps (splash volume and upsampling) are performed for a specific scenario, it can be used efficiently to experiment with different sets of values for parameters (as shown in Fig. 3.16)

Because our splash volume is computed by the interactions between a closed mesh and the input liquid, we must extend slightly the interface of the updated SDF by an epsilon value to gather the part of the coarse liquid in the vicinity of the moving object. For

Example	**Simulation time**			**Simulation**			
	Low Res.	High Res.	Ours	Advection	Seeding Tracers	Artificial Pressure	Implicit Waves
Spherical drop	0.9	7.0	0.3	14%	18%	23%	46%
Ball drop	1.0	6.5	0.4	14%	16%	23%	47%
Hulk's wrath	3.1	9.3	0.5	16%	23%	21%	40%
Speedboat ride	3.5	10.5	0.7	17%	13%	19%	52%
Fractal fruits	5.2	102.7	0.9	16%	14%	18%	52%
River	2.6	11.2	0.8	15%	12%	17%	55%

Table 3.2. Discretization statistics and simulation computation times breakdown. The computation times are expressed in seconds per frame.

example, we used an epsilon equal to 0.2 (twice the particle separation) for the *Fractal fruits* (Fig. 4.1) and the *Ball drop* scenarios (Fig. 3.11) with a domain size of 8 × 5 × 8. For the scenario of *Hulk's wrath* (Fig. 3.13), epsilon is 0.5 for dimensions of 10 × 5 × 3. We extend the SDF to ensure that the resulting splash volume will contain most of the particles affected by this induced interaction. The *Speedboat ride* scenario (Fig. 3.10) is a

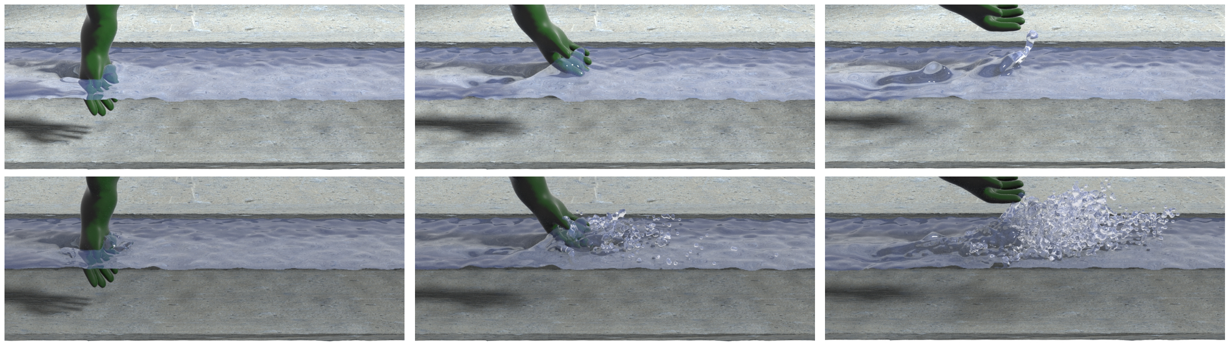


Figure 3.13. *Hulk's wrath* is a good example of our approach with solid-liquid interactions. (top) The splash is almost nonexistent in the low-resolution example. (bottom) Our method manages to generate a realistic splashing liquid out of this very coarse liquid.

good example to push the limits of our method. It would be reasonable to think that our method could be even more costly than a traditional FLIP method if the resulting splash volume were generated over the whole simulation domain. As shown in the *Speedboat* *ride* scenario, the path followed by the animated boat passes through most of the domain. Therefore, the evaluated splash volume at the end of the animation is indeed as large as the surface covering the whole liquid. However, even with a difficult scenario like this one, we managed to keep increasing the splash details while reducing significantly (compared to the equivalent high resolution) the associated computation time per frame. The criteria for upsampling are key to avoid oversampling at non-visually-interesting areas. As evidenced with the low-resolution example of this scenario, the boat does not generate much detail except when tight turns occur. From the input coarse resolution, our score based on the tendency of a particle to reach the surface helps to wisely upsample the underlying cells of the discretized splash volume. As shown in Table 3.3, the *Speedboat ride* example was more

Example	**Number of particles**			**Pre-computation***	
	Low Res.	High Res.	Upsampled Res.	Splash Volume	Upsampling
Spherical drop	95k	1.2M	115k	2.3	1.4
Ball drop	100k	0.9M	115k	2.5	1.5
Hulk's wrath	225k	1.1M	150k	3.6	1.8
Speedboat ride	280k	2.3M	300k	34.2	2.5
Fractal fruits	400k	85.4M	450k	6.2	3.5
River	250k	10.6M	350k	3.3	5.2

Table 3.3. Discretization statistics and pre-computation times breakdown. Our method adds upsampled particles to the low-resolution particles. * The pre-computation steps (in seconds) are excluded from the simulation times (Table 3.2) since they are computed only once before simulating.

costly in pre-computation steps because we had to compute the volume based on the whole animation (500 frames). In contrast, the *Hulk's wrath* splash volume example was very fast to compute, considering that the hand was in contact with the liquid for just a few frames of the animation. In such cases, we were able to stop early the volume pre-computation step. The *Fractal fruits* (Fig. 4.1) example successfully demonstrates the capability and scalability of our post-processing approach. Once the required pre-computation steps are performed (total of 9.7 s to compute and upsample the splash volume for 200 frames), our method faithfully reproduces realistic and parameterizable splash details with less than one second per frame at runtime. According to this demonstration, our approach achieves a speedup factor of more than 100× over the high-resolution reference (see Table 3.2). Furthermore, the advantages of our approach also become apparent when simulating with different sets of values for parameters in order to obtain the desired result (e.g., as shown with Fig. 3.16). We also experiment with our approach in a more dynamic environment. The *River* scenario (available in the accompanying video) demonstrates the flexibility of our approach on a high-velocity flow. As explained in § 3.4.1, we compute the splash volume from the first collision between the upsampling and colliding volumes (i.e., Ω^{up} and Ω^{c}). However, in this case, since the initial velocity of the upsampling volume is fairly high, we process the upsampling step at the first collision frame.

3.8.3. Liquid-Liquid Interactions

We also provide examples of liquid-liquid interactions. This type of scenario can produce an adequate splash volume by separating the input liquid into different volumes. For instance, with the *Spherical drop* scenario (Fig. 3.9), we divided at initialization the domain into two parts: the sphere and the liquid in the container. The sphere is handled similarly to a mesh in the sense that we pre-compute an SDF based on the related coarse particles. The resolution used for computing the level set needs to be high enough to capture and separate at the initial state (i.e., at the first frame or at emission) the input coarse liquid into distinct volumes. We used a resolution of 128^3 in all our examples and it has proved to be sufficient for our purposes.

	Fig. 3.9(a)	**Fig. 3.9(b)**	**[103]**	**[110]**	**Ours**
# spray / splash particles	-	-	49k	262k	115k
Time per iteration (ms)	900	7000	2	248	80

Table 3.4. Computation times comparison with state-of-the-art methods and ground truth (i.e., high resolution) on an example as shown in Fig. 3.12.

3.8.4. Comparison with Spray Particles

We compared our approach with state-of-the-art spray particle methods. In order to present a fair comparison, only the contributions associated with spray particles are exposed. As shown in Fig. 3.12(b), Chentanez and Müller [103] introduce a very diffuse model to simulate spray particles. Since they target real-time applications, their spray model is added on top of a heightfield fluid, preventing particle-particle interactions. More similarly to ours, Ihmsen et al. [110] present a fully procedural spray model adapted for particle-based liquids. Their approach is capable of generating convincing diffuse behaviors, the motion of the spray particles is solely influenced by external forces and gravity (similar to our upsampled particles flagged as ballistics). This is where our approach and the purpose of tracers stand out. The localized forces applied by the tracers reproduce a better surface-tension behavior (Fig. 3.12(d)) as compared to the spray particles proposed by Ihmsen et al. [110] (Fig. 3.12(c)). Moreover, our implicit linear wave model generates smooth swirls at the base of the crown splash, introducing better interactions between our upsampled model and the coarse input liquid, as opposed to Ihmsen et al. [110].

3.8.5. Integration with Hybrid-DFSPH

To demonstrate that our approach works with most particle-based methods, we implemented our splash volume and upsampling method in a hybrid DFSPH model [131] (results are shown in the accompanying video). Although the velocity field can be estimated from the particle velocities, using the one provided from a hybrid model is even better. Our approach is completely independent of the input architecture. Nevertheless, we had to constraint the upsampling step to focus on regions with the splash volume bounded by the input narrow band of particles. We experiment with different strategies but doing so prevents us from generating artifacts deep under the liquid interface (i.e., in the Eulerian portion of the simulation).

3.8.6. Comparison with Uniform Upsampling

As shown in Fig. 3.14, our upsampling method faithfully preserves the small-scale details at the top of the crown splash (image on the right) while a uniform upsampling (image on the

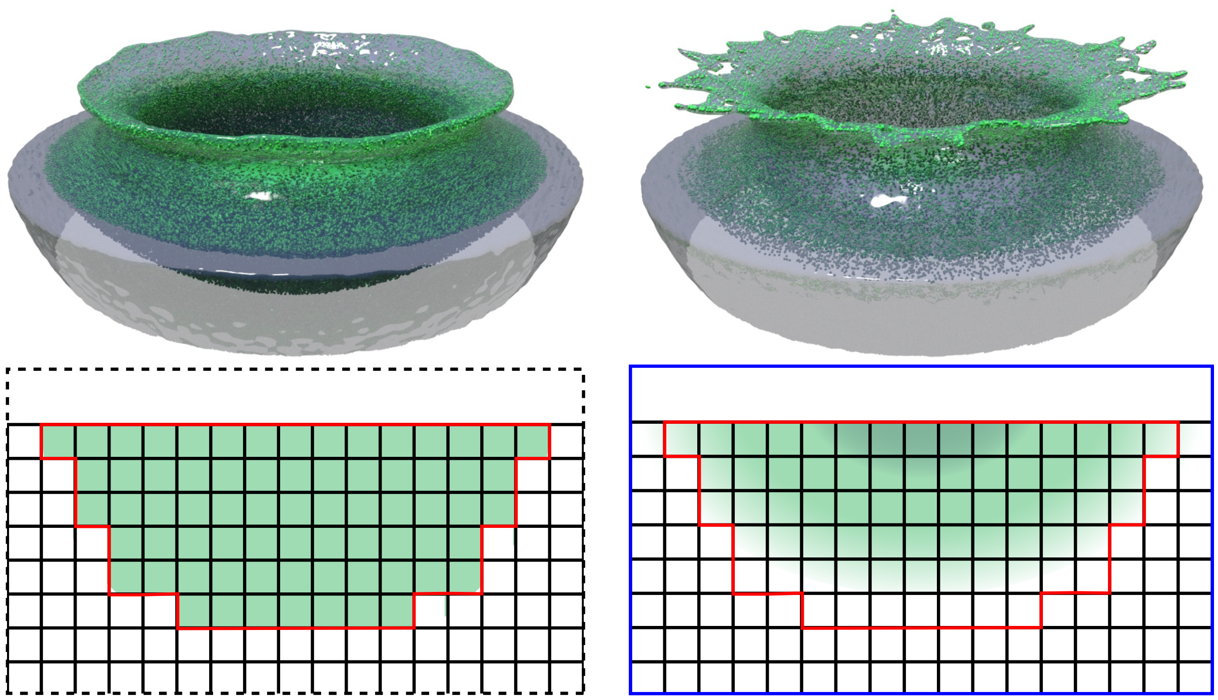

(a) Uniform upsampling (b) Emerge-tendency upsampling

Figure 3.14. This crown splash example shows a comparison with a naive (a) uniform upsampling method and (b) our upsampling method (right). (Bottom) The density of particles per cell is shown in green inside the splash volume (red).

left) causes clustering of particles at the base of the splash, in regions where velocity magnitudes tend to be zero over time. Adding particles to these regions will hardly contribute to increase details. Our upsampling method offers a more effective way to upsample and distribute particles within the splash volume. As shown in the graph of Fig. 3.15, our upsampling method presents a distribution trend similar to the one of the Maxwell-Boltzmann distribution. As stated by Mandl [121] and in reference to the Maxwell-Boltzmann statistics, the density of particles highly depends on the distribution of velocities inside the fluid. By oversampling particles in the regions identified by our method, we ensure to solely increase particle density to bring out fine details dissipated and bounded by the input resolution. Also, since our method highly upsample a subdomain of the input simulation, preserving such a distribution is critical for detail enhancement and to avoid undesired particle clumping. As expected, our upsampling method preserves a high density of particles in cells with velocities close to $\|\bar{\vec{u}}\|$ (average velocity norm). Low densities of particles are kept in cells with

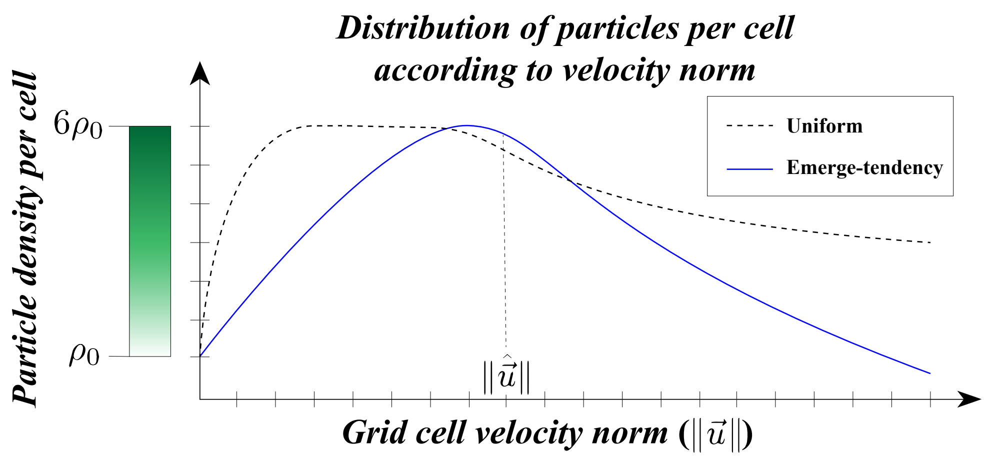


Figure 3.15. This graph plots the density against the velocity distribution exclusively within the splash volume of the scenario shown in Fig. 3.14. ρ_0 corresponds to the number of particles per cell as used by the input coarse liquid; $6\rho_0$ represents the maximum number of upsampled particles used in this examples (i.e., $s_{up} = 6$). Also note that the dash plot (uniform upsampling) indicates many more particles in total (upsampled) than the continuous blue plot (emerge-tendency upsampling).

low velocities and very high velocities (way above average). The low densities of particles in high-velocity cells correspond to highly diffuse particles (e.g., such as small droplets).

3.9. Conclusion and Future Work

We have presented a flexible and physics-motivated post-processing approach to generate detailed and realistic splash behaviors from a coarser particle-based input liquid. Our approach is divided into different steps. A splash volume is computed and upsampled by analyzing the surrounding environment and by determining where sudden disturbances occur. The upsampled particles advected alongside the coarse liquid provide a practical way to enrich portions of liquid that could not otherwise capture such details. We also provided a novel way to affect splashing particles through tracers and localized artificial pressure corrections. Finally, our efficient implicit wave model offers a controllable method to process interactions between upsampled splash particles and input particles. As noted by the statistics presented in Table 3.2, even if we compare the computation times by adding the simulation of a low-resolution liquid to our upsampling method, we still manage to outperform the reference high-resolution simulations. In addition, once the pre-calculation steps are completed, our method allows a user, without any limitation, to adjust the level of detail almost interactively.

Nevertheless, our method suffers from a few limitations. First of all, precomputing the splash volume once for the whole animation prevents us from upsampling at different time intervals. In other words, highly dynamic scenarios would require an adaptive upsampling method to address frequent changes in the regions of interest through time. Our approach is currently designed to better suit scenarios in which the upsampled portion is initially static. An evolving 4D splash volume would be an interesting direction for future work. By tracking these splash volumes temporally, we could improve our upsampling method to cover complex interactions. Secondly and along the same lines, while our method offers a controllable way to improve the apparent resolution of splashing liquids, some high-resolution scenarios would hardly benefit from its current state. For example, simulating the collision between a high-velocity volume of liquid and a complex geometry (e.g., firehose shooting water on a highly detailed armadillo) would barely benefit from our proposed upsampling approach.

Because of the flexibility of our approach, we plan to explore more art-directable methods to use controls similar to our tracers as a modeling tool for splashes and turbulent flows. It would be also relevant to improve the steps of upsampling and tracer seeding at different scales through machine learning techniques. State-of-the-art neural network approaches could improve the prediction on seeding locations for better results. It could also provide an efficient way to upsample particles based on learned high-resolution examples, and result in even more similar splashes.

Acknowledgments

This work was supported and funded by Mitacs, Prompt, NSERC, Digital District Canada (formerly Mokko Studio), and the University of Montreal. We would like to thank Digital District for providing resources and insightful comments during our research. We would also like to thank Side Effects Software for providing us with Houdini educational licenses.

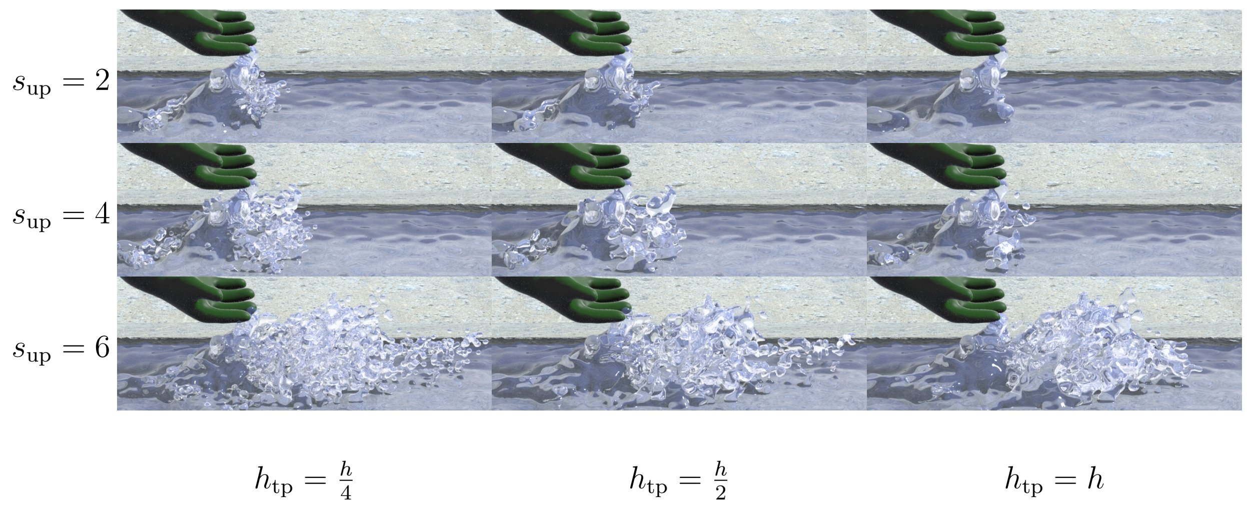


Figure 3.16. With the *Hulk's wrath* example, we show the influence of the scaling factor s_{up} on the number of upsampled particles per cell (rows) and the tracers support radius h_{tp} (columns) on the appearance of the liquid.

References

[93] Mridul Aanjaneya, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Trans. on Graphics (TOG)*, 36(4):140, 2017.

[94] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J Guibas. Adaptively sampled particle fluids. 26(3):48–55, 2007.

[95] Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Trans. on Visualization and Computer Graphics*, 18(8):1202–1214, 2012.

[96] Ryoichi Ando, Nils Thürey, and Chris Wojtan. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. on Graphics (TOG)*, 32(4):103, 2013.

[97] Ryoichi Ando and Reiji Tsuruno. A particle-based method for preserving fluid sheets. In *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 7–16, 2011.

[98] Jan Bender and Dan Koschier. Divergence-free SPH for incompressible and viscous fluids. *IEEE Trans. on Visualization and Computer Graphics*, 23(3):1193–1206, 2017.

[99] Landon Boyd and Robert Bridson. MultiFLIP for energetic two-phase fluid simulation. *ACM Trans. on Graphics (TOG)*, 31(2):16, 2012.

[100] Robert Bridson. Fast Poisson disk sampling in arbitrary dimensions. In *SIGGRAPH sketches*, page 22, 2007.

[101] Robert Bridson. *Fluid Simulation for Computer Graphics.* CRC Press, 2015.

[102] José A Canabal, David Miraut, Nils Thuerey, Theodore Kim, Javier Portilla, and Miguel A Otaduy. Dispersion kernels for water wave simulation. *ACM Trans. on Graphics (TOG)*, 35(6):202, 2016.

[103] Nuttapong Chentanez and Matthias Müller. Real-time simulation of large bodies of water with small scale details. In *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 197–206, 2010.

[104] Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Trans. on Graphics (TOG)*, 36(4):69, 2017.

[105] Hilko Cords. Moving with the flow: Wave particles in flowing liquids. *Journal of WSCG*, pages 145–152, 2008.

[106] Mathieu Desbrun and Marie-Paule Cani. *Space-time adaptive simulation of highly deformable substances.* PhD thesis, INRIA, 1999.

[107] Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. Narrow band FLIP for liquid simulations. *Computer Graphics Forum*, 35(2):225–232, 2016.

[108] Christopher Jon Horvath and Barbara Solenthaler. Mass preserving multi-scale SPH. Technical Report 13–04, Pixar Animation Studios, 2013.

[109] Hikaru Ibayashi, Chris Wojtan, Nils Thuerey, Takeo Igarashi, and Ryoichi Ando. Simulating liquids on dynamically warping grids. *IEEE transactions on visualization and computer graphics*, 26(6):2288–2302, 2018.

[110] Markus Ihmsen, Nadir Akinci, Gizem Akinci, and Matthias Teschner. Unified spray, foam and air bubbles for particle-based fluids. *The Visual Computer*, 28(6-8):669–677, 2012.

[111] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible SPH. *IEEE Trans. on Visualization and Computer Graphics*, 20(3):426–435, 2014.

[112] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH fluids in computer graphics. In *Eurographics - State of the Art Reports*, 2014.

[113] Stefan Jeschke and Chris Wojtan. Water wave packets. *ACM Trans. on Graphics (TOG)*, 36(4):103, 2017.

[114] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 49–57, 1990.

[115] Janghee Kim, Deukhyun Cha, Byungjoon Chang, Bonki Koo, and Insung Ihm. Practical animation of turbulent splashing water. In *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 335–344, 2006.

[116] Theodore Kim, Jerry Tessendorf, and Nils Thuerey. Closest point turbulence for liquid surfaces. *ACM Trans. on Graphics (TOG)*, 32(2):15, 2013.

[117] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. *ACM Trans. on Graphics (TOG)*, 27(3):50, 2008.

[118] Lubor Ladicky, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Trans. on Graphics (TOG)*, 34(6):199, 2015.

[119] Frank Losasso, Jerry Talton, Nipun Kwatra, and Ronald Fedkiw. Two-way coupled SPH and particle level set fluid simulation. *IEEE Trans. on Visualization and Computer Graphics*, 14(4):797–804, 2008.

[120] Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. on Graphics (TOG)*, 32(4):104, 2013.

[121] Franz Mandl. *Statistical Physics (2nd Edition).* John Wiley & Sons, 2008.

[122] Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim, and Derek Nowrouzezahrai. Surface turbulence for particle-based liquid simulations. *ACM Trans. on Graphics (TOG)*, 34(6):202, 2015.

[123] Viorel Mihalef, Dimitris Metaxas, and Mark Sussman. Simulation of two-phase flow with sub-scale droplet and bubble effects. *Computer Graphics Forum*, 28(2):229–238, 2009.

[124] Joseph J Monaghan. SPH without a tensile instability. *Journal of Computational Physics*, 159(2):290–311, 2000.

[125] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proc. Symposium on Computer Animation*, pages 154–159. ACM SIGGRAPH/Eurographics, 2003.

[126] Matthias Müller-Fischer. Fast water simulation for games using height fields. In *Proc. of the Game Developer's Conference*, page 24, 2008.

[127] Rahul Narain, Jason Sewall, Mark Carlson, and Ming C Lin. Fast animation of turbulence using energy transport and procedural synthesis. *ACM Trans. on Graphics (TOG)*, 27(5):166, 2008.

[128] Jens Orthmann and Andreas Kolb. Temporal blending for adaptive SPH. 31(8):2436–2449, 2012.

[129] Tobias Pfaff, Nils Thuerey, Andrew Selle, and Markus Gross. Synthetic turbulence using artificial boundary layers. *ACM Trans. on Graphics (TOG)*, 28(5):121, 2009.

[130] Bruno Roy, Eric Paquette, and Pierre Poulin. Particle upsampling as a flexible post-processing approach to increase details in animations of splashing liquids. *Computers & Graphics*, 88:57–69, 2020.

[131] Bruno Roy and Pierre Poulin. A hybrid Eulerian-DFSPH scheme for efficient surface band liquid simulation. *Computers & Graphics*, 77:194–204, 2018.

[132] Takahiro Sato, Christopher Wojtan, Nils Thuerey, Takeo Igarashi, and Ryoichi Ando. Extended narrow band FLIP for liquid simulations. In *Computer Graphics Forum*, volume 37, pages 169–177, 2018.

[133] Hagit Schechter and Robert Bridson. Evolving sub-grid turbulence for smoke animation. In *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 1–7, 2008.

[134] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. on Graphics (TOG)*, 24(3):910–914, 2005.

[135] Xuqiang Shao, Zhong Zhou, Jinsong Zhang, and Wei Wu. Realistic and stable simulation of turbulent details behind objects in smoothed-particle hydrodynamics fluids. *Computer Animation and Virtual*

Worlds, 26(1):79–94, 2015.

[136] SideFX. Houdini (version 16.5). `https://www.sidefx.com/products/houdini-fx/`, 2016.

[137] Barbara Solenthaler and Markus Gross. Two-scale particle simulation. *ACM Trans. on Graphics (TOG)*, 30(4):81, 2011.

[138] Barbara Solenthaler and Renato Pajarola. Density contrast SPH interfaces. In *Proc. Symposium on Computer Animation*, pages 211–218. ACM SIGGRAPH/Eurographics, 2008.

[139] Barbara Solenthaler and Renato Pajarola. Predictive-corrective incompressible SPH. *ACM Trans. on Graphics (TOG)*, 28(3):40, 2009.

[140] Tsunemi Takahashi, Hiroko Fujii, Atsushi Kunimatsu, Kazuhiro Hiwada, Takahiro Saito, Ken Tanaka, and Heihachi Ueki. Realistic animation of fluid with splash and foam. *Computer Graphics Forum*, 22(3):391–400, 2003.

[141] Jerry Tessendorf. Vertical derivative math for iWave. Technical report, Clemson University, 2008.

[142] Nils Thuerey, Theodore Kim, and Tobias Pfaff. Turbulent fluids. In *ACM SIGGRAPH 2013 Courses*, page 6, 2013.

[143] Kiwon Um, Xiangyu Hu, and Nils Thuerey. Perceptual evaluation of liquid simulation methods. *ACM Trans. on Graphics (TOG)*, 36(4):143, 2017.

[144] Kiwon Um, Xiangyu Hu, and Nils Thuerey. Liquid splash modeling with neural networks. *Computer Graphics Forum*, 37(8):171–182, 2018.

[145] Chang-Bo Wang, Qiang Zhang, Fan-Long Kong, and Hong Qin. Hybrid particle-grid fluid animation with enhanced details. *The Visual Computer*, 29(9):937–947, 2013.

[146] Huamin Wang, Gavin Miller, and Greg Turk. Solving general shallow wave equations on surfaces. In *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 229–238, 2007.

[147] Xiangyun Xiao, Cheng Yang, and Xubo Yang. Adaptive learning-based projection method for smoke simulation. *Computer Animation and Virtual Worlds*, 29(3-4):e1837, 2018.

[148] Sheng Yang, Xiaowei He, Huamin Wang, Sheng Li, Guoping Wang, Enhua Wu, and Kun Zhou. Enriching SPH simulation by approximate capillary waves. In *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 29–36, 2016.

[149] Zhi Yuan, Ye Zhao, and Fan Chen. Incorporating stochastic turbulence in particle-based fluid simulation. *The Visual Computer*, 28(5):435–444, 2012.

[150] Cem Yuksel, Donald H House, and John Keyser. Wave particles. *ACM Trans. on Graphics (TOG)*, 26(3):99, 2007.

[151] Bo Zhu, Wenlong Lu, Matthew Cong, Byungmoon Kim, and Ronald Fedkiw. A new grid structure for domain extension. *ACM Trans. on Graphics (TOG)*, 32(4):63, 2013.

[152] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. on Graphics (TOG)*, 24(3):965–972, 2005.

Chapter 4

Neural UpFlow: A Scene Flow Learning Approach to Increase the Apparent Resolution of Particle-Based Liquids

This third paper introduces a deep learning approach to enhance the level of detail of particle-based liquids. Our network allows us to generalize high-resolution details for liquids and to express them as displacements. To generate the displacements, the proposed architecture learns differences between positions based on the neighborhoods of the particles. We also use an adapted interpolation method to infer the magnitude of the deformations between pairs of liquids at low and high resolutions. The predicted displacements are then expressed as a Lagrangian motion to advect the particles inferring dissipated details bounded by the input resolution. In complement to our work, we also provide a framework using this same interpolation method to augment particle-based liquid datasets for multi-resolution training purposes.

Publication

This paper [177] was published in Proceedings of the ACM in Computer Graphics and Interactive Techniques (PACMCGIT), proceedings of the ACM SIGGRAPH/Eurographics

Symposium on Computer Animation 2021 (SCA), in September 2021. It has been reformatted for this thesis. Roy was involved in the formulation and numerical analysis of this approach. He was also responsible for the design, implementation, training, and evaluation of the approach. The first draft of the published paper was written by Roy and improved by Poulin and Paquette prior and during the submission process.

Neural UpFlow: A Scene Flow Learning Approach to Increase the Apparent Resolution of Particle-Based Liquids

BRUNO ROY, *Université de Montréal and Autodesk*
PIERRE POULIN, *Université de Montréal*
ERIC PAQUETTE, *École de technologie supérieure*

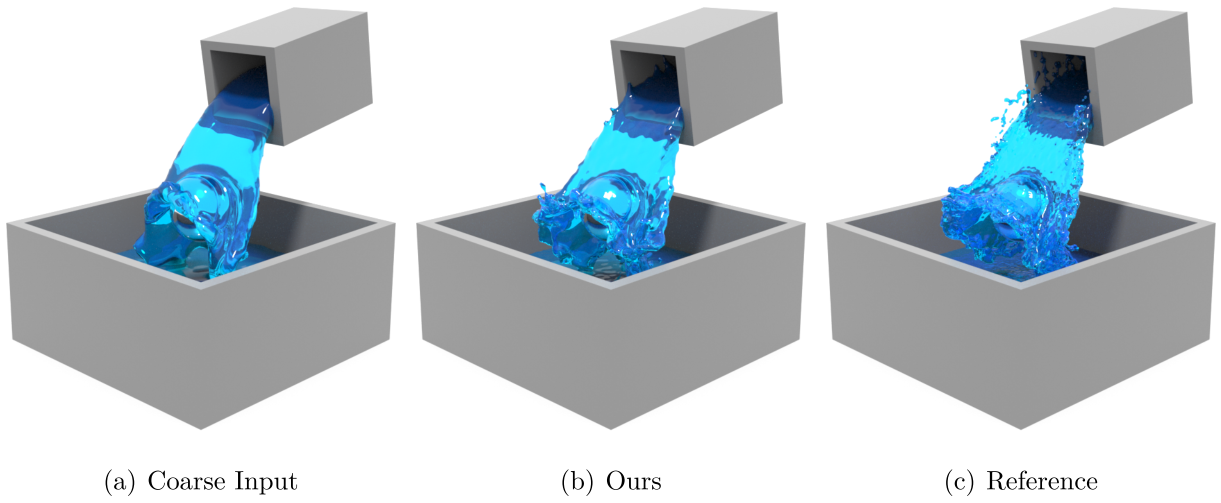


Figure 4.1. Our learning deformation field reproduced (b) most of the small- and large-scale details from the high-resolution ground truth (c) using solely the low-resolution input example (a). Particles are added and displaced by our network to generate *plausible* details of up-resed animations at a fraction of a high-resolution simulation cost.

Abstract

We present a novel up-resing technique for generating high-resolution liquids based on scene flow estimation using deep neural networks. Our approach infers and synthesizes small- and large-scale details solely from a low-resolution particle-based liquid simulation. The proposed network leverages neighborhood contributions to encode inherent liquid properties throughout convolutions. We also propose a particle-based approach to interpolate between liquids generated from varying simulation discretizations using a state-of-the-art bidirectional optical flow solver method for fluids in addition with a novel key-event topological alignment constraint. In conjunction with the neighborhood contributions, our loss formulation allows

the inference model throughout epochs to reward important differences in regard to significant gaps in simulation discretizations. Even when applied in an untested simulation setup, our approach is able to generate plausible high-resolution details. Using this interpolation approach and the predicted displacements, our approach combines the input liquid properties with the predicted motion to infer semi-Lagrangian advection. We furthermore showcase how the proposed interpolation approach can facilitate generating large simulation datasets with a subset of initial condition parameters.
Keywords: Fluid Simulation, Particle-Based Liquid, Deformation Field, Optical Flow, Up-Resing, Machine Learning, Deep Neural Network

4.1. Introduction

Machine learning approaches have been rising in popularity in the last few years due to their accessibility and versatility. These learning approaches have been adapted for many applications in media and entertainment areas, and more specifically in graphics, such as animation [154], motion capture [182], style transfer [186], 2D-to-3D sketching techniques [162], physically-based rendering [159], texture synthesis [164], fluid simulations [165], and so on.

In recent years, the use of machine learning on fluids has considerably reduced the emphasis on computationally expensive numerical methods while taking advantage of the existing knowledge in the field. So far, machine learning techniques have been used to improve numerical solvers [168, 194, 181], to compute fluid features [184, 189, 165, 183], and to infer visual details [171, 161]. However, very few of these methods have directly addressed the notion of improving the apparent resolution of *particle*-based fluids, particularly in the context of learning on unstructured data. Lately though, researchers in the computer vision community [172, 173] have studied irregular data, such as point clouds, and introduced ways to directly process them as inputs for 3D classification and segmentation tasks. Moreover, their work has been recently extended [169, 187] to learn on temporal motions of point clouds such as defined using scene flow estimation methods. Considering structural similarities between point clouds and particle systems, we demonstrate in this paper that using scene flows expressed as Lagrangian motions on particles is promising to infer fine and controllable details on fluids.

The work most similar in spirit to ours is the one by Prantl et al. [171]. They propose to leverage neural networks to learn precomputed deformations of liquid surfaces. While we partially share their goal in encoding space-time deformations for liquids, there are several noticeable differences. First, we target particles as opposed to signed distance functions (SDF) to provide greater flexibility on the method used to apply deformations. We show that applying deformations directly on particles before generating a surface facilitates bringing out fine details, often depending on the discretization. Second, using a neighborhood embedding layer, our network does not require multiple learning phases to encode realistic high-resolution liquid features. In addition, we use a deformation weight obtained from the work of Thuerey [180] to guide (i.e., through our deformation-aware loss function) and speed up convergence.

In this paper, we propose an adapted deep neural network architecture that combines a recent scene flow machine learning method with particle neighborhood interactions expressed as fluid simulation properties. Our network exploits particle neighborhoods inspired by hybrid liquid simulation methods to encode hierarchical features of the particle-based simulations during the convolution operations. We show that we are able to preserve important liquid features at every level of convolution. Our loss function also includes a coefficient to weigh important local features while reducing convergence time. Finally, we introduce an advection scheme that allows us to displace the particles while taking the actual low-resolution motion into account. This same advection scheme is also used to artificially grow our existing multi-resolution simulation dataset using an adapted interpolation method. With this comprehensive learning pipeline, our approach is able of accurately reproducing high-resolution details using solely a very coarse particle-based liquid. To summarize, the main contributions of our work are:

- an adapted deep neural network architecture using particle neighborhoods and a deformation-aware loss function reducing the inference noise encoded during convolutional operations,
- an advection scheme that transposes a scene flow into Lagrangian motion using the input velocity field,
- an interpolation scheme for matching key events topological changes between simulations at different resolutions, and

- a data augmentation framework that artificially grows an existing particle-based simulation dataset.

4.2. Related Work

Multi-Scale and Procedural Methods. In the last decade or so, several methods have been proposed to enhance the apparent resolution of liquid simulations. While procedural methods were more practical for decoupling simulations from discretization, multi-scale methods were particularly efficient to process independently surface details and separate them from coarse volumes of liquid. Using a resampling strategy to segregate multiple layers of the simulation data (e.g., particles) has been revisited multiple times. Adaptive models were proposed to dynamically adjust the size (and contributions on forces) of particles using split-and-merge operations within the simulation loop [155, 190]. While these methods share the same goal in spirit, Winchenbach et al. [190] leverage the fundamental basis of Smoothed Particle Hydrodynamics (SPH) methods. Their proposed scheme enables defining with precision particle masses to improve stability while preserving small-scale details. Similarly, Solenthaler and Gross [179] introduced an adaptive method to couple a dual particle-layer scheme using two distinct particle resolutions. The method of Winchenbach et al. [190] was also extended to improve spatial adaptivity by introducing a continuous objective function as a refinement scheme for SPH [191].

A method focusing on preserving thin sheets was introduced by Ando et al. [156] allowing them to adapt the resolution of particles in *Fluid Implicit Particle (FLIP)* simulations. Later, narrow-band methods were introduced to get the best of both Eulerian and Lagrangian schemes by expressing the liquid surface with particles and the coarser volume with a grid [163]. The method was first introduced to *FLIP* given its semi-Lagrangian nature. The method was then extended by Sato et al. [178] to process arbitrary locations (as opposed to exclusively the liquid surface), identifying where particles are needed to refine the appearance of the surface. The same idea was revisited and adapted to SPH methods [174, 160, 176]. Other adaptive methods were also proposed to exploit spatially adaptive structures to capture different simulation scales [157, 153].

Meanwhile, procedural methods have also been introduced as a refinement scheme by generating surface points through wave simulations. As noted by Kim et al. [166], high-frequency details are not necessarily coupled to the coarse simulation and can be independently generated directly onto the animated mesh. In comparison to the work of Kim et al. [166], which was applied in an Eulerian setup (i.e., solely based on the corresponding SDF and the velocity field), Mercier et al. [170] proposed a Lagrangian up-res method to increase the apparent resolution of *FLIP* with a secondary surface wave simulation. Recently, another procedural method was introduced focusing solely on splashing liquid details [175]. As we focus on neural networks (NN) to enrich the liquid surface, we will not compare our work with procedural methods. Although we share the similar goal of improving the apparent resolution of the liquid surface at a fraction of the cost and computational time, our approach offers the capabilities to synthesize plausible details because it is trained on real fluid simulations.

Machine Learning for Fluids. Leveraging machine learning methods for fluids was first introduced in graphics by Ladický et al. [168]. They demonstrated the inference of SPH forces to approximate Lagrangian positions and velocities using regression forests. Along the same line of ideas, a similar method was lately proposed to apply neural networks to Eulerian methods [194]. More recently, NNs were combined with numerical solvers to model diffuse particle statistics for the generation of splashes [183]. Convolutional neural networks (CNN) were also widely used, but instead to predict Eulerian simulation properties. Thompson et al. [181] introduced a method exploiting CNNs to speed up existing solvers. They proposed a CNN-based architecture as a preconditioner for the pressure projection step. CNNs were also used to learn flow descriptors to visually enhance coarse smoke simulations using precomputed patches [161]. In a context where synthetic and plausible samples can be generated on-demand, using generative adversarial networks (GAN) seemed suitable for fluids. GANs were initially used as a super-resolution technique to enhance temporally coherent details of smoke simulations directly in image-space [193]. Meanwhile, Kim et al. [165] proposed another generative network for precomputing solution spaces for smoke and liquid flows. Super-resolution was later revisited for fluids, enabling an LSTM-based architecture to predict the pressure correction and generate physically plausible high-frequency details [188]. Due to their sequential properties, LSTM-based layers were later used by Wiewel et al. [189] to predict changes of the pressure field for multiple subsequent timesteps. Another work on convolutional networks for Lagrangian fluids has been presented by Ummenhofer et al. [184].

They proposed a way to express particle neighborhoods using spatial convolutions as differentiable operators. In contrast with these methods, our network learns to encode differences between resolutions as opposed as to map fluid features using similarities.

Recently, Prant et al. [171] proposed a stacked neural network architecture allowing them to learn space-time deformations for interactive liquids. Although we share the same goal of precomputing simulation data of the liquid surface, our work focuses on the learning in a Lagrangian setup for up-resing purposes. Moreover, in comparison to the work of Prant et al. [171], in which two stacked NNs are used to learn deformations, we propose a simpler and more efficient NN architecture by using a deformation coefficient directly in our objective function and by exploiting the particle neighborhoods to guide the convolution downsampling layers.

4.3. Method

Given consecutive frames of an input animation of a liquid, our approach predicts Lagrangian deformations to infer high-resolution details and behavior. We propose an adapted scene flow learning architecture, that we call *FluidFlowNet* (*FFNet*), to determine Lagrangian motions on particle-based liquids. Our learning architecture is inspired from *FlowNet3D* [169]; it encodes displacements between two unstructured and unordered particle-based liquids simulated at different resolutions using solely their positions and velocities as input (§ 4.5.1). Our dataset is composed of over one thousand randomly generated pairs of liquid simulation scenarios in which each pair is composed of two simulations using the same initial parameters but at different resolutions: one simulation at a coarse resolution and the other at a high resolution (§ 4.4.2). Similarly to the work of Thuerey [180], we compute a mapping correspondence between the two liquids using an optical flow solve on a 4D volume. However, in our case, the resulting optical flow is used to capture the deformations between pairs of multi-resolution simulations guiding our loss function. We use the multi-resolution liquid pairs as a training set to encode a deformation using our *FFNet* model. Finally, the inferred deformation field obtained from our learning pipeline is transposed into Lagrangian displacements and used as up-resing operators on the low-resolution particle-based liquid (§ 4.6). With these displacements, we encode the missing details and behavior bounded from the input low-resolution simulation. In addition, a post-processing step refines the particles' positions combining the low-resolution input velocities with the applied displacements while reducing the inference noise. Fig. 4.2 shows an overview of our up-resing pipeline for liquids. In this paper, we use bold lowercase symbols for vectors and bold uppercase symbols for matrices. For clarity in Fig. 4.2, we use dotted symbols for the Lagrangian properties. For instance, the velocity of a particle would be noted as $\dot{\boldsymbol{u}}$.

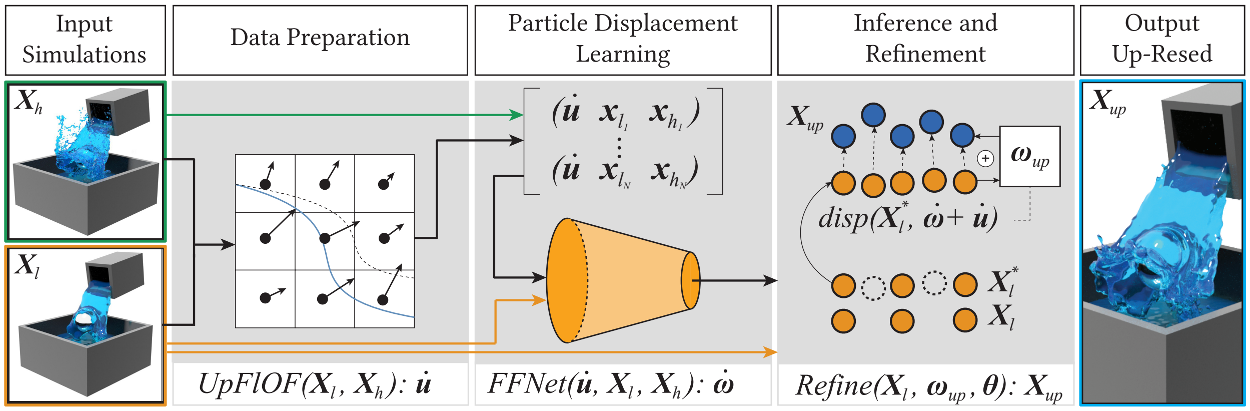

Input Simulations
Data Preparation
Particle Displacement Learning
Inference and Refinement
Output Up-Resed
X_h
X_l
$(\dot{u}\ x_{l_1}\ x_{h_1})$
$(\dot{u}\ x_{l_N}\ x_{h_N})$
X_{up}
ω_{up}
$disp(X_l^*, \dot{\omega}+\dot{u})$
X_l^*
X_l
X_{up}
$UpFlOF(X_l, X_h): \dot{u}$
$FFNet(\dot{u}, X_l, X_h): \dot{\omega}$
$Refine(X_l, \omega_{up}, \theta): X_{up}$

Figure 4.2. Overview of our up-resing pipeline. The input data used during the training phase is prepared using pairs of simulations at low (in orange outline) and high resolution (in green outline), which is also used to generate a deformation field. Our network encodes particle displacements using differences between neighboring particles as a scene flow. These predicted displacements are then refined and applied directly to the particles to infer details (result in blue outline).

Preparing Simulation Data. The dataset is built in three stages: (1) generating many pairs of liquid animation scenarios (i.e., low-resolution and high-resolution) using different subsets of initial conditions, (2) producing a deformation field using an optical flow solve on generated 4D volumes (used later in our loss function), and (3) augmenting artificially the number of training samples using the deformation field to interpolate new variants of simulated scenarios.

Learning Particle-Based Deformations. We use a neural network to learn and encode Lagrangian deformations between low- and high-resolution particle-based liquids. The proposed *FFNet* exploits the output of the optical flow solve to weigh each deformation between the input local features. The input features to our network are then expressed as the particle positions for the low- and high-resolution liquids.

Inference and Refinement. Because we apply the deformations directly on particles, our approach firstly requires to upsample the particles close to the liquid interface before inference. Also, we generate our results using *Narrow Band FLIP* liquids to reduce the memory footprint since the deformations are mainly modeled according to surface particles. Finally, we deploy a refinement step at inference to account for generalization errors and to adjust the final particle motion according to the input velocity field.

4.4. Inter-Resolution Liquid Interpolation

In this section, we adapt an algorithm originally from Thuerey [180] to interpolate between a low- and a high-resolution liquid. We refer to *inter-resolution* as an interpolation performed between a low-resolution simulation and a high-resolution simulation from an SDF point of view. We use this interpolation approach in our proposed pipeline for two reasons: preparing the data for training with our neural network, and combining Eulerian deformations with the particle motions. Using an Eulerian deformation factor along with a particle-based deformation field ($\dot{\boldsymbol{u}}$ in Fig. 4.2), our approach interpolates between resolutions directly on the particles. We are expressing these deformations in a Lagrangian manner to increase small-scale details and to offer a more intuitive control to infer deformations to particles. As highlighted in Fig. 4.3, applying deformations directly on the particles allows a more faithful reproduction of fine details of the interpolation target (e.g., the sharp edges of the cube). In addition, we have observed that our approach gives a more realistic liquid appearance while preserving certain visual characteristics such as surface tension. In order to provide a fair comparison, both interpolation methods were performed using the same grid resolution.

Our motivation in applying displacements directly to the particles is to better reflect the small-scale deformations while not being strictly bounded to the resolution of an underlying grid. An Eulerian deformation is quite successful in detecting smooth large-scale motions, but its inherent regularization prevents it from matching fine surface details at the cost of additional correction steps. Inspired by the method proposed by Thuerey [180], we use a 4D optical solve to interpolate between input scenarios using the same initial conditions, but at different resolutions (low and high resolutions).

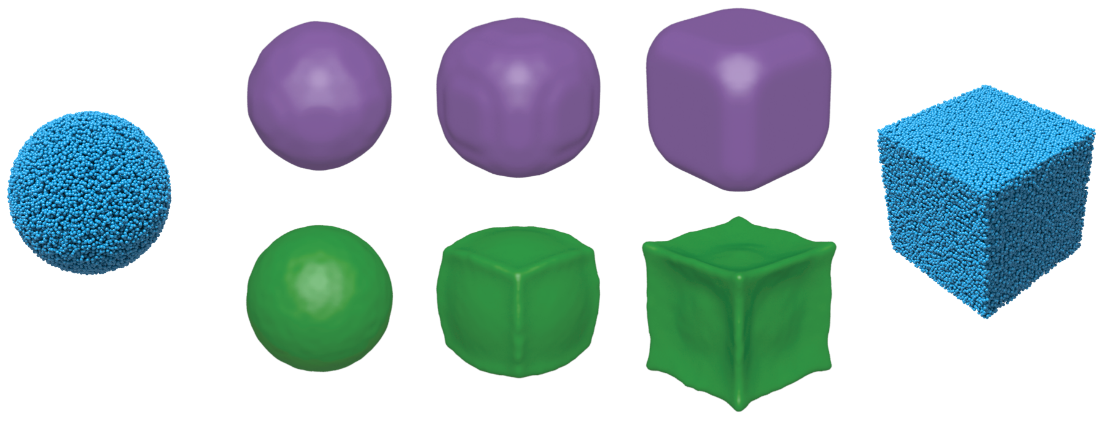

(a) Input (b) Interpolation (c) Target

Figure 4.3. A simple case where a sphere is morphed at three different steps (b) into a cube: comparing our interpolation approach applied directly on particles (bottom in green) with a state-of-the-art interpolation method [180] applied on the corresponding SDF (top in purple).

4.4.1. Up-Resing Optical Solve

Inter-Resolution Optical Flow. We employ an approach similar to the work of Thuerey [180], that we briefly summarize in the following, as we propose a slightly altered form of it for learning purposes. Although our goal is also to compute a deformation field, we adapt Thuerey's method to interpolate a low-resolution input into a higher resolution one, using the same initial conditions for both inputs. The generated deformation field is used with the particles of each simulation (i.e., low- and high-resolution) as input features of the training set for our neural network (*FFNet*).

Given a pair of corresponding surfaces (i.e., generated SDF $\boldsymbol{\Phi}$ from the input particles) using two distinct simulation resolutions (i.e., number of particles and grid resolution), the optical flow solve is expressed as a weighted sum of energy terms to be minimized to compute deformation field $\boldsymbol{u}$:

$$\min_{\boldsymbol{u}} \; E_d(\boldsymbol{u}) + \beta_S E_{\text{smooth}}(\boldsymbol{u}) + \beta_T E_{\text{Tikhonov}}(\boldsymbol{u}), \tag{4.1}$$

where the first term corresponds to the information related to the SDF (i.e., occupancy proportion values). The second and third terms are respectively the smoothness and Tikhonov regularizers. The smoothness term E_{smooth} penalizes non-smooth solutions, and the Tikhonov term penalizes vectors with large magnitudes. The discretized minimization of Eq. 4.1 yields a system of linear equations $\boldsymbol{A}_{\text{UpOF}}\boldsymbol{u} = \boldsymbol{b}$ where the first term $\boldsymbol{A}_{\text{UpOF}}$ corresponds to the discretized energy terms:

$$\boldsymbol{A}_{\text{UpOF}} = \nabla\boldsymbol{\Phi}_h{}^T\nabla\boldsymbol{\Phi}_h + \beta_S\sum_j \mathbf{L}_j + \beta_T\mathbf{I}, \tag{4.2}$$

and where the terms are respectively the discrete spatial gradient squared, the smoothness (using the discrete Laplacian $\mathbf{L}$), and the Tikhonov regularizers. Finally, we express the right-hand side $\mathbf{b}$ of the linear system as $[\nabla\boldsymbol{\Phi}_h]^T\,\boldsymbol{\Phi}_\delta$ where $\boldsymbol{\Phi}_\delta$ is the finite difference between the input surfaces $\boldsymbol{\Phi}_h$ and $\boldsymbol{\Phi}_l$. Solving the linear system for $\boldsymbol{u}$ gives us the up-res deformation field $\boldsymbol{u}_{\text{up}}$. Up to this point, our approach differs only from that of Thuerey [180] by its preparation and application to the input data as we use it between simulation pairs of varying resolutions (see Algo. 7 for details).

Key-Event Alignment. As we focus on the differences between variable resolution inputs, we propose an additional and novel key-event alignment term directly within the optical flow solve to constrain the solution according to specific surface topological changes. The motivation is to align the deformations between inputs simulated at different resolutions to capture and match certain key events in time. To do so, we define a soft constraint expressed as a penalty term $\mathbf{D}$ detecting cells (using the grid of the SDF $\boldsymbol{\Phi}$) with topological changes. Since our method is already applying the optical solve on the SDF, we decided to use the

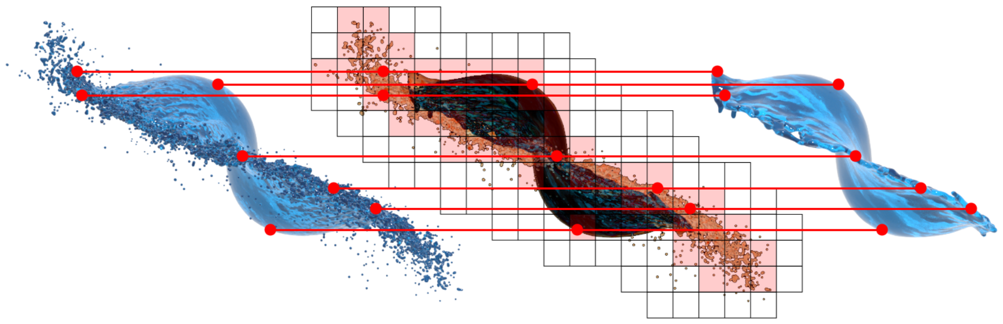


Figure 4.4. As highlighted in this illustration, the key-event constraint term allows matching surface points between (right image) low- and (left image) high-resolution liquid surfaces. The complex cells are marked with a red background (middle image); the feature surface points are identified and connected using red points and lines.

complex cell tests [192] to determine which cells are too complex to be represented with piecewise linear isosurfaces [185]. Each cell complexity c_i (0 or 1) in $\mathbf{D}$ is weighted using a proportional coefficient $d_{\boldsymbol{x}_i \to \boldsymbol{x}_k}$ based on its 4D Euclidean distance to the closest key surface point $\boldsymbol{x}_k$:

$$d_i = \frac{1}{d_{\boldsymbol{x}_i \to \boldsymbol{x}_k}} c_i. \tag{4.3}$$

The closest key surface point $\boldsymbol{x}_k$ is computed using the same feature point extraction method as used with point cloud registration. The feature points are selected using the local mean curvature approximation μ and compared to the domain distribution. We keep the feature points above a certain threshold with respect to their distribution $\mu + \alpha\rho$. The coefficient α is used to control the number of feature surface points preserved. The Euclidean distance $d_{\boldsymbol{x}_i \to \boldsymbol{x}_k}$ is computed using the first step of the *Iterative Closest Point (ICP)* algorithm which is used to determine the closest 4D key surface point $\boldsymbol{x}_k$ for each surface point at position $\boldsymbol{x}_i$ (Fig. 4.4). Using the proportional coefficient d_i, we can then rewrite Eq. 4.2 as follows:

$$\boldsymbol{A}_{\text{UpOF}} = \nabla\boldsymbol{\Phi}_h^T \nabla\boldsymbol{\Phi}_h + \beta_S \sum_j \mathbf{L}_j + \beta_T \mathbf{I} + \mathbf{D}. \tag{4.4}$$

As shown in Fig. 4.5, small-scale artifacts arise when trying to interpolate between two liquids having major differences bounded by their resolution. By aligning the deformations $\boldsymbol{u}_{\text{up}}$ using key-event feature surface points, we improve the fidelity (i.e., as close as possible to the high-resolution ground truth) of deformation on the low-resolution input.

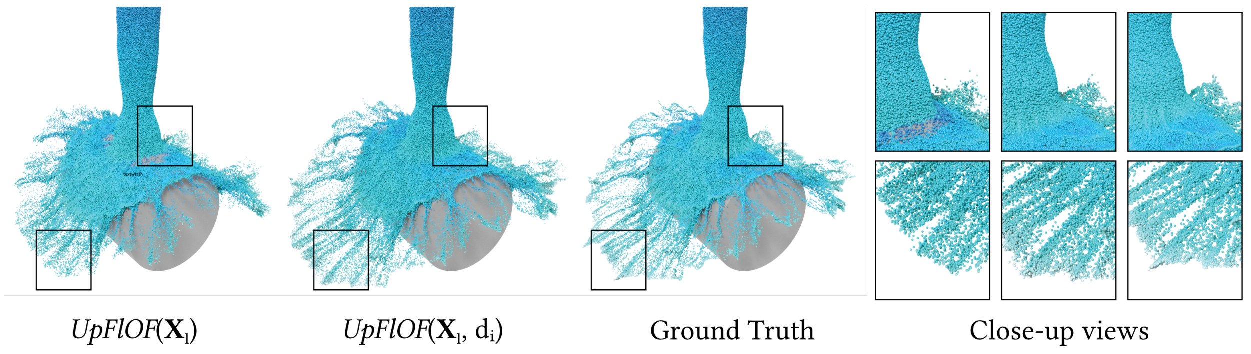


Figure 4.5. Comparing the deformation precision of $\boldsymbol{u}_{\text{up}}$ when applied to a low-resolution surface using our key-event alignment term.

The motivation behind using the proportional coefficient d_i with complex cells is to influence the deformation solution $\boldsymbol{u}_{\text{up}}$ according to key cells (i.e., cells containing key points) defining a correspondence between two simulations. However, the discretization of a simulation may produce significant differences that can make it difficult to determine a match. Knowing that generating a match between these key events in some scenarios would be challenging, we opted for a soft constraint even though the solution would sometimes be partially satisfied (as opposed to enforcing a hard constraint on Eq. 4.4).

4.4.2. Dataset and Data Augmentation

Our dataset is composed of pairs of simulated *Fluid Implicit Particle (FLIP)* liquids using initial conditions from a parameter matrix $\mathbf{\Theta}$. We would like to note that nothing in our particle-based method is limited to FLIP, but FLIP and Narrow Band FLIP are commonly used and efficient, which facilitated the generation of our datasets. As previously stated, each pair is composed of two simulated liquids: one at a low resolution and the other at a higher resolution. We adjusted the liquid resolution by changing the particle spacing (i.e., particle density per cell) and the underlying grid resolution. For all of the presented examples, we used a particle separation (ps) of 0.02 and 0.005, and a grid scaling factor (i.e., used to compute the cell size with the particle spacing) (gs) of 2.0 and 1.2, respectively for low- and high-resolution liquid samples. We define a pair of sample liquids as $P_i = \{\boldsymbol{X}_l, \boldsymbol{X}_h\}_i$ where each sample $\boldsymbol{X}_i$, regardless of its resolution, is expressed as a set of particle coordinates $\boldsymbol{x}$.

The simulated liquids are generated using selected initial conditions taken from a matrix (as shown in Algo. 6). We refer to each of these initial parameters using a subscript to our parameter matrix $\mathbf{\Theta} \in \mathbb{R}^{n \times m}$ where n is the number of parameters, and m the number of initial values for each parameter (e.g., obstacle shape type $\mathrm{o_{st}}$, obstacle position $\mathrm{x_o}$, emitter position $\mathrm{x_{em}}$, container dimensions cd). As an example, the notation $\mathbf{\Theta}_{\mathrm{x_{em}}}$ would be for an emitter position of a sample simulation. The resolution parameters, particle spacing ps, and grid scale gs are not included in the matrix because they are fixed for all pairs of simulated liquids. For our dataset purpose, the emitter velocity is computed with respect to the emitter position in order to be oriented toward an existing obstacle and/or container. The types of shapes o_{st} can be used as static boundaries or as a liquid initial shape. We present in Appendix 4.11 a small subset of generated samples for the training set.

Data Augmentation. As shown in the second half of Algo. 6, we use data augmentation on the simulated liquids to artificially grow the number of samples within our dataset. In

Algorithm 6: Pseudo-code for generating and augmenting our training dataset.

```
Input: $\left.\begin{array}{l} \mathrm{o_{st}}\text{: obstacle shape type(6 different shapes)} \\ \mathrm{x_o}\text{: obstacle position(}\mathrm{x_o} \in \mathbb{R}^3\text{)} \\ \mathrm{x_{em}}\text{: emitter position(}\mathrm{x_{em}} \in \mathbb{R}^3\text{)} \\ \text{cd: container dimensions(cd} \in \mathbb{R}^3\text{)} \end{array}\right\} = \boldsymbol{\Theta}$

$P = \emptyset$
for $\forall \boldsymbol{\Theta}_i \in \boldsymbol{\Theta}$ do                          /* for each parameter set $\boldsymbol{\Theta}_i$ */
    $\boldsymbol{X}_l = \text{simulate}(\boldsymbol{\Theta}_i, ps = 0.02, gs = 2.0)$          /* FLIP solver */
    $\boldsymbol{X}_h = \text{simulate}(\boldsymbol{\Theta}_i, ps = 0.005, gs = 1.2)$         /* FLIP solver */
    $P \cup \{\boldsymbol{X}_l, \boldsymbol{X}_h\}$
$P^* = P$
for $\forall P_i \in P$ do                  /* for each pair of simulated liquids $P_i$ */
    $P_j = \text{randomize}(\{P_j | j \neq i, j \in 0 \leq j \leq |P|\})$
    $\{\boldsymbol{X}_l, \boldsymbol{X}_h\}_i, \{\boldsymbol{X}_l, \boldsymbol{X}_h\}_j = P_i, P_j$
    $\boldsymbol{X}_l^* = \mathit{UpFlOF}(\{\boldsymbol{X}_l\}_i, \{\boldsymbol{X}_l\}_j, \boldsymbol{\Phi}(\{\boldsymbol{X}_l\}_i), \boldsymbol{\Phi}(\{\boldsymbol{X}_l\}_j), \alpha_{\text{FlOF}} = 0.5)$
    $\boldsymbol{X}_h^* = \mathit{UpFlOF}(\{\boldsymbol{X}_h\}_i, \{\boldsymbol{X}_h\}_j, \boldsymbol{\Phi}(\{\boldsymbol{X}_h\}_i), \boldsymbol{\Phi}(\{\boldsymbol{X}_h\}_j), \alpha_{\text{FlOF}} = 0.5)$
    $P^* \cup \{\boldsymbol{X}_l^*, \boldsymbol{X}_h^*\}$
```

this regard, we use the original *FlOF* algorithm as proposed by Thuerey [180] to interpolate liquids between precomputed initial conditions. We first randomly select each simulation pair P_i with a pair P_j (where $i \neq j$) to generate a new pair of liquids. Then, we extract for each pair their associated particle coordinates $\boldsymbol{X}$ as inputs to the interpolation method. The interpolated liquid is obtained by computing a bidirectional optical flow $\boldsymbol{u}_\omega$ applied on the surface $\boldsymbol{\Phi}(\boldsymbol{X})$. We use $\alpha_{\text{FlOF}} = 0.5$ to generate an in-between simulation for each interpolated liquid (see the *UpFlOF* method in Algo. 7).

Even though the resulting liquids generated by interpolation are inevitably not physically accurate, they still provide significant additional data points allowing us to improve the generalization rate when it comes to predicting complex high-resolution behaviors.

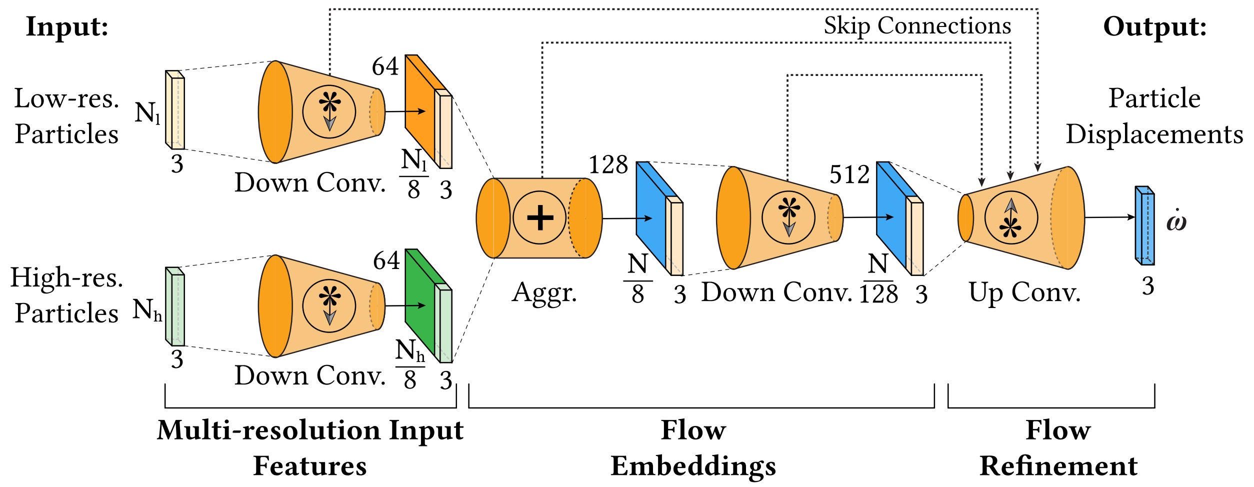


Figure 4.6. The proposed network architecture. Given a pair of input liquid simulations, the particle deformation network focuses on encoding the particle displacement between low- and high-resolution particle-based simulations guided by the outputs from the *UpFlOF* approach. The dimensions at each convolution layer is expressed as a portion of the input resolution (N: number of upsampled particles from the input) by 3 (number of components of the input feature). The number of local features grows as we progress down in the downsampling convolutions (e.g., 32, 64, 128, and so on).

4.5. Scene Flow Learning for Lagrangian Deformation on Particles

In this section, we will further detail the network architecture of our proposed approach (§ 4.5.1) and our neighborhood-based loss function for particle-based liquids (§ 4.5.2).

4.5.1. Network Architecture

As previously mentioned in § 4.3, our proposed learning architecture (Fig. 4.6), which we call *FFNet*, is an adaptation of the *FlowNet3D* method [169] in order to successfully capture the inherent and latent liquid properties. By giving additional cues of the underlying liquids, our learning approach converges faster to a more precise displacement solution.

Inspired by *FlowNet3D*, our network architecture is divided into three main modules: multi-resolution input features, flow embedding, and flow refinement. We will describe how we adapted each module in the following.

Multi-Resolution Input Features. In this first part of our learning pipeline, we use the local neighborhoods to estimate the convolution operators, as proposed with the *PointNet++*

architecture [173]. Since a traditional convolution does not work with unstructured data points such as a particle set, using a downsampling approach based on the neighborhood provides a suitable way to encode these local features.

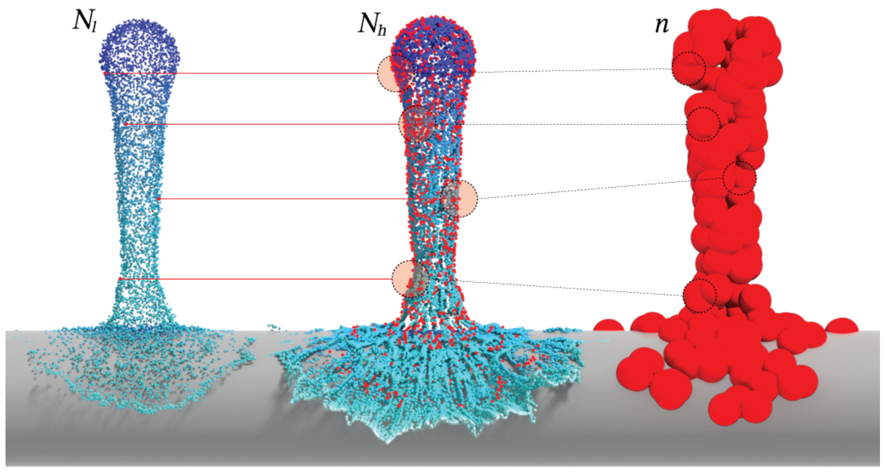


Figure 4.7. At each iteration for our network, a correspondence map is generated between the input pairs of multi-resolution simulations $\boldsymbol{X}_l$ and $\boldsymbol{X}_h$ consisting of respectively N_l and N_h particles. These n correspondences are expressed as representative neighborhood for each level of convolution.

As shown in Fig. 4.7, a convolution layer downsamples an input liquid with N particles into n neighborhoods. Each particle $p_i = \{\boldsymbol{x}_i, \boldsymbol{f}_i\}$, where $\boldsymbol{x}_i$ is the $\mathbb{R}^3$ coordinates of particle i and $\boldsymbol{f}_i$ its associated local features generated at each layer of our network. Using the SPH neighborhood computation [158], we consider the contributions of each particle p_i in its neighborhood using weighted averages. As opposed to computing each weighted average at the position of a particle, we compute the contributions at the center of each downsampled neighborhood n_j. The weighted local features $\boldsymbol{f}_i^*$ for a downsampled neighborhood are then computed as follows:

$$\boldsymbol{f}_i^*(\boldsymbol{x}) = \boldsymbol{f}_i|\boldsymbol{x} - \bar{\boldsymbol{x}}_{n_j}|, \tag{4.5}$$

where $\boldsymbol{x}$ is the center coordinates of neighborhood n_j, and $\bar{\boldsymbol{x}}_{n_j}$ is the weighted average of the coordinates of particles in that neighborhood:

$$\bar{\boldsymbol{x}}_{n_j} = \sum_{i \in n_j} w_i \boldsymbol{x}_i. \tag{4.6}$$

The weights for each neighborhood n_j are computed using the same smooth kernel function introduced by Zhu and Bridson [195]:

$$w_i(x) = \frac{k(|\boldsymbol{x} - \boldsymbol{x}_i|R^{-1})}{\sum_j k(|\boldsymbol{x} - \boldsymbol{x}_j|R^{-1})}, \tag{4.7}$$

where $k(s) = \max(0,(1 - s^2)^3)$ for a smooth transition between neighbor contributions, and the neighborhood radius R is equal to twice the particle separation used to generate the input simulation. We express the weighted features as the importance level of each local feature during the convolution phases.

We have noticed throughout experiments that using the weighted average to estimate the local features for each downsampled neighborhood n_j improved the recall when predicting large-scale flows. Finally, we use an element-wise max pooling operator **MAX** on a nonlinear approximation using a multi-layer perceptron (MLP) on the weighted $\boldsymbol{f}_i^*(x)$ as follows:

$$\boldsymbol{f}_{n_j}^* = \underset{\{i \| \|\boldsymbol{x}_i - \boldsymbol{x}'_{n_j}\| \leq R\}}{\mathbf{MAX}} \{h(\boldsymbol{f}_i^*, \boldsymbol{x}_i - \boldsymbol{x}'_{n_j})\}, \tag{4.8}$$

where h is the nonlinear MLP function, $\boldsymbol{x}'_{n_j}$ is the center coordinate of the neighborhood n_j, and R the same radius as used in Eq. 4.7.

Fluid Flow Embedding. Once the weighted local features are computed for an input pair (i.e., for both low and high resolutions), the two local features $f_{n_j}^*$ and $g_{n_j}^*$, respectively generated from the low-resolution input $\boldsymbol{X}_l$ and the high-resolution input $\boldsymbol{X}_h$, are combined before applying convolution to training samples.

Similarly to creating the multi-resolution input features, we use the same number of neighborhoods to describe each particle-based input liquid pair, even if they are discretized differently. That way, our network can define the flow embeddings using the same neighborhood center coordinates $\boldsymbol{x}_{n_j}$ for a simulation pair alongside the weighted local features and the original particle coordinates. We use the same weighted function $w_i(x)$ as input to the nonlinear MLP function h to aggregate each neighbor contribution prior to being max pooled. We express each particle embedding as:

$$e_i = \underset{\{i \| \|\boldsymbol{x}_i - \boldsymbol{x}'_j\| \leq R\}}{\mathbf{MAX}} \{h(\boldsymbol{f}_{n_j}^* \oplus \boldsymbol{g}_{n_j}^*, \{\boldsymbol{x}_l\}_i - \{\boldsymbol{x}_h\}_i)\}, \tag{4.9}$$

where $\boldsymbol{f}_{n_j}^*$ aggregates the input features of the downsampled neighborhood n_j from the low-resolution particle-based liquid, and $\boldsymbol{g}_{n_j}^*$ the input features of the downsampled neighborhood

n_j from the high-resolution one. We perform a few additional convolutions to spatially smooth out the corresponding features with respect to each particle of each neighborhood.

Flow Refinement. Finally, as the last step of the network, we focus the learning on propagating the embedded features e_i from the downsampled neighborhood n_j to the original low-resolution input particles N_l. We use upsampling convolution operations to learn how to project weighted features back to the input particle coordinates. We have noticed throughout experiments that propagating the weighted features from the embedding layer improved the ability of the network to recover nonlinear features specific to fluids, such as vorticity.

We show the precision of our refinement step on downsampled features (Fig. 4.8) as opposed to solely using a symmetric MLP function on these. A last and dense regression layer is used after the upsampling convolutions to project back the scene flow prediction in $\mathbb{R}^3$ (i.e., without using an activation function). Also note that as shown in Fig. 4.6, we use skip connections between downsampling and upsampling layers to infer multi-scale feature learning.

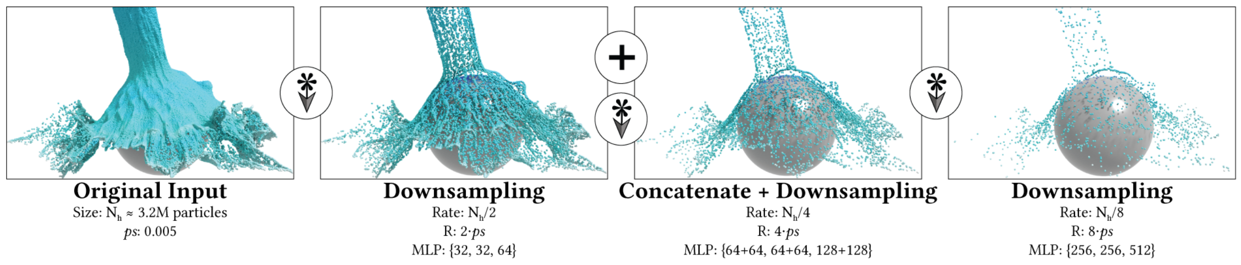


Figure 4.8. Example of a cascade of downsampling convolution operations applied on a high-resolution particle-based liquid.

4.5.2. Deformation-Aware Loss Function

As explained previously, we are using pairs of particle-based simulations as input samples. Our training loss on these samples is evolving throughout epochs to capture the differences in displacements between simulations using the same initial condition parameters, but at different discretizations. We express our loss function as L_1 metrics to encode absolute differences between the low-resolution and high-resolution samples:

$$\mathcal{L}_{\text{up}} = \frac{1}{n_j} \sum_{i}^{n_j} \|\boldsymbol{\omega}_i - \boldsymbol{\omega}_i^*\|_1 + \lambda_{n_j} \|\boldsymbol{\omega}_i^{\leftarrow} - \boldsymbol{\omega}_i\|_1, \tag{4.10}$$

where $\boldsymbol{\omega}_i$ is the predicted displacement and $\boldsymbol{\omega}_i^*$ is the ground truth. Interestingly, the cycle-consistency regularization term $\|\boldsymbol{\omega}_i^{\leftarrow} - \boldsymbol{\omega}_i\|_1$ (in Eq. 4.10) acts as a penalization constraint to enforce the bidirectionality of the resulting scene flow for displacements. In other words, this term enforces that the forward and backflow flows (i.e., displacement between the predicted particle coordinates and the displaced one $\boldsymbol{\omega}_i^{\leftarrow}$ using $\boldsymbol{\omega}_i^*$) closely match each other. Also, we have noticed empirically that adding an adaptive weight λ_{n_j} to adjust the neighborhood n_j contributions (as opposed to using $\lambda = 1$ for every neighborhood) has improved significantly the convergence rate throughout iterations. We compute λ_{n_j} using the normalized magnitude of the deformation initially computed with the *UpFlOF* algorithm (Algo. 7). From our analysis comparing different training experiments, we have observed that using such an adaptive weight was allowing emphasis on encoding features where the main differences between discretized simulations were occurring within neighborhoods. Throughout the experiments, we also observed that the proposed adaptive weight improved (in some of the results presented) the reconstruction of small complex details, as we were able to train through more iterations without overfitting the resulting generalization model.

4.6. Deformation Inference and Refinement

In the following sections, we dive into the final steps of applying the predicted displacements onto the input particle-based liquid. First, we apply the displacements taking into account the existing velocities of the input particles. Then, a refinement step is performed to reduce the displacement noise generated by the inference.

4.6.1. Applying the Displacement on Particles

In this section, we explain the steps performed by our approach to combine the input velocity with the predicted Lagrangian displacements $\dot{\boldsymbol{\omega}}$. Firstly, we upsample the input narrow band $\boldsymbol{X}$ of depth d_b to ensure that the density of the upsampled particle set $\boldsymbol{X}'$ is high enough to capture the small-scale deformations generated by our resulting deformation field. Once the predicted displacements are generated, we transfer them to a velocity field $\boldsymbol{u}_\omega$.

The cell size of $\boldsymbol{u}_\omega$ is adjusted with respect to the number of particles per cell to prevent undesirable gaps in the liquid during motion, as suggest by Zhu and Bridson [195]. We resample the velocity field in a MAC grid $\boldsymbol{u}_{\text{MAC}}$ that will then be updated by extrapolation within a distance d_{MAC} outside the SDF $\boldsymbol{\Phi}'_l$ in order to fully cover the simulation domain of the upsampled input particles $\boldsymbol{X}'$. Using the updated $\boldsymbol{u}_{\mathrm{MAC}}$, we can now update our displacements to compensate for the actual input motion. That way, our approach can infer Lagrangian displacement regardless of the input motion. The MAC grid $\boldsymbol{u}'_{\mathrm{MAC}}$ is weighted according to the magnitudes of the resampled velocity field $\hat{\boldsymbol{u}}_\omega$ and added to the current velocity field $\hat{\boldsymbol{u}}_\omega$. Finally, we advect the particles in a grid (as with *FLIP*) using $\hat{\boldsymbol{u}}'_\omega$ on the upsampled particles $\boldsymbol{X}'$ at each time step. Although our approach requires switching back and forth between Eulerian and Lagrangian deformations, we have empirically noticed that our model learns better to preserve fine details on the surface, as opposed to learning to deform these solely in an Eulerian manner. An overview summarizing these steps is presented with Algo. 7 (with *isInference()* equal to *true*).

4.6.2. Upsampling and Reducing Surface Noise

We have noticed during our experiments that surface noise was introduced at the inference step when compared to the ground truth. In order to validate the source of this noise, we have investigated on the resulting deformations before combining them with the input motion (i.e., exclusively based on $\dot{\boldsymbol{\omega}}$). Similar to regression methods with point clouds (such as used with scene flow), this regression noise was appearing in the downsampling operations during the learning phase. As suggested by Liu et al. [169], performing multiple passes of inference on randomized resampled samples using average predictions improved the results, but at the cost of losing some of the small-scale liquid behavior mostly perceptible during motion.

Since our goal is to focus the deformation close to the surface, performing a regression on the whole particle-based fluid seemed like an inadequate solution. As a matter of fact, the resulting noise came mostly from particles emerging from deep below the surface. We then determined that a resampling approach as used in *Narrow Band FLIP* [163] would be more suitable to convey where to resample in order to prevent displacing particles deep in the liquid (as opposed to randomly resampling). We ended up performing multiple passes of inference using varying depths d defining the particle band thickness.

As shown in Fig. 4.9, we came up with a fair tradeoff throughout several experiments. For our purpose, this tradeoff is purely qualitative and might differ for other types of simulations. An error (between the displacement and the ground truth) smaller than 0.1 is barely

Algorithm 7: *UpFlOF* - Pseudo-code for inferred semi-Lagrangian advection and data augmentation.

```
Input: $\boldsymbol{X}$: set of particles
       $\boldsymbol{u}_{\mathrm{MAC}}$: low-res. MAC velocity grid
       $d_{\mathrm{MAC}}$: extrapolation distance (2 cells)
       $d_b$: depth of narrow band (2-3 cells)
       $r_h$: grid resolution larger than the input
$\boldsymbol{\Phi}_l$ = computeSDF($\boldsymbol{X}$)
$\boldsymbol{X}'$ = resampleNarrowBand($\boldsymbol{X}, d_b$)
if isInference() then                                   /* for predicting displacements */
    $\dot{\boldsymbol{\omega}}$ = predict($\boldsymbol{X}$)                              /* see Fig. 4.6 */
    $\boldsymbol{u}_\omega$ = transferToGrid($\dot{\boldsymbol{\omega}}$)
    $\boldsymbol{\Phi}_l' = \boldsymbol{\Phi}_l$
else                                                    /* for data augmentation */
    $\boldsymbol{\Phi}_l'$ = upscaleInterpolate($\boldsymbol{\Phi}_l, r_h$)
    $\boldsymbol{u}_\omega$ = FlOF($\boldsymbol{\Phi}_l', \boldsymbol{\Phi}_h, \alpha_{\mathrm{FlOF}} = 1$)           /* see Algo. 6 in [180] */
$\hat{\boldsymbol{u}}_\omega$ = resampleOFtoMAC($\boldsymbol{u}_\omega, \boldsymbol{u}_{\mathrm{MAC}}$)
$\boldsymbol{u}_{\mathrm{MAC}}$ = extrapolate($\boldsymbol{u}_{\mathrm{MAC}}, \boldsymbol{\Phi}_l', d_{\mathrm{MAC}}$)
$\boldsymbol{u}_{\mathrm{MAC}}' = \|\hat{\boldsymbol{u}}_\omega\| \boldsymbol{u}_{\mathrm{MAC}}$
$\hat{\boldsymbol{u}}_\omega' = \hat{\boldsymbol{u}}_\omega + \boldsymbol{u}_{\mathrm{MAC}}'$                                 /* inject input motion */
$\boldsymbol{X}_{\mathrm{advect}}$ = advectInGrid($\boldsymbol{X}', \hat{\boldsymbol{u}}_\omega', t$)                 /* $t = t + \triangle t$ */
```

noticeable (especially in motion). Also, as exposed in Fig. 4.9 (rightmost image generated after 12 iterations), performing more than 6 inference iterations is diluting the high-resolution features and ultimately converging back to the low-resolution input (i.e., fading out the displacements). The final displacement obtained is then averaged over all the iterations.

4.7. Results

In the following, we discuss the training experiments and expose details of the corresponding datasets. We also demonstrate on multiple examples the capabilities of our approach to increase the apparent resolution using solely coarse particle-based simulations as input. We refer the reader to the supplemental video for the corresponding animations.

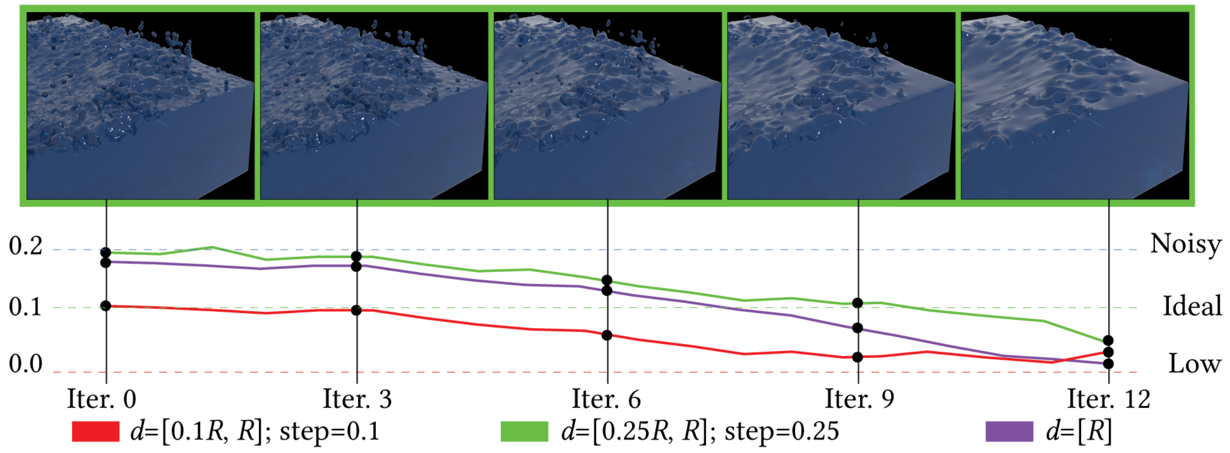


Figure 4.9. We compare the influence of varying upsampling distances d over the surface noise (expressed as an error on the vertical axis) generated throughout the inference iterations. The images corresponding to the evolving level of noise (i.e., from iteration 1 to 12) of the green curve is presented on the top of the chart.

4.7.1. Training and Inference

Datasets and Augmentation. The simulations in our dataset have been generated using the *Bifröst*® fluid solver for *Maya*®. The simulations for both resolutions are computed using varying parameters as exposed in Algo. 6. We have divided the datasets into three categories referred to as *Colliding*, *Shape*, and *Container* (see Table 4.1 for further details). These datasets are covering several simple and specific simulation cases to improve inference generalization. The *Colliding* dataset contains simulations in which an emission source hits a single collision shape (see Fig. 4.19 in Appendix 4.11). The *Shape* dataset contains simulations in which a single liquid volume (initialized with different shapes) falls into an empty container. The *Container* dataset contains simulations in which an emission source pours into a liquid container. For all these datasets, we define the breadth of simulation scenarios used for training by enumerating all combinations from a fixed set of obstacle shapes o_{st}, obstacle positions $\boldsymbol{x}_{\mathrm{o}}$, position of emission sources $\boldsymbol{x}_{\mathrm{em}}$, and dimensions of container cd. The shapes were determined to cover different shapes including sharp edges (cube), curved faces (sphere), and a mixture of both (cylinder).

In order to artificially grow each of these datasets for training, we interpolate each simulation pair using Thuerey's method [180] on shape's SDF, shape position, and emission

Dataset	Simul. res. (low/high)		Augment.	# simul.
	# particles	Veloc. grid	factor	pairs
Colliding	0.30M / 2.4M	96^3 / 192^3	$\times 2$	432
Shape	0.15M / 1.5M	96^3 / 192^3	$\times 2$	432
Container	0.40M / 3.2M	128^3 / 256^3	$\times 8$	432

Table 4.1. Datasets of simulation pairs used as training and validation sets. We also present the augmentation factor (Augment. factor) used to grow our datasets prior to training.

position. On both *Colliding* and *Shape* datasets, we interpolated with weight $\alpha_{\text{FlOF}} = 0.5$, giving us an augmentation factor of $\times 2$. We used more interpolation weights on the *Container* dataset since it originally consisted of fewer simulations. Each simulation pair of the *Container* dataset was interpolated at $\alpha_{\text{FlOF}} \in \{0.25, 0.50, 0.75\}$ giving us an augmentation factor of $\times 8$. Finally, we used a data split of 90%:10% respectively for training and validation sets. The test set was composed of new and significantly different simulation setups as enumerated in Table 4.2. We also provide a breakdown of the evaluation times of the presented examples in Appendix 4.10.

Performance Analysis. We have tested our approach on a variety of simulation setups to validate its robustness when generalizing within unknown initial conditions. As shown in Table 4.2, we tested our learning model on input simulations of a fairly coarse resolution. The simulation time to generate the coarse simulation in Maya® is presented because that simulation data is used as input to our network for inference. As previously mentioned, since we are mostly interested in surface details, we opted for the *Narrow Band FLIP* method to generate the particle-based liquids used for both training and testing inference. The computation times presented are expressed in seconds and computed at each frame of simulation and inference. The computation times of our *FFNet* network at inference are also presented. An interesting aspect when looking at these is that they are only slightly influenced by the input complexity. Another benefit of using the neighborhood convolutions to downsample multi-resolution particle-based simulations is that the evaluation times at inference turns out almost constant and decoupled from the input resolution as we progress down the convolution layers.

Example	**# part.**	**Grid res.**	**Simul.**	***FFNet* eval.**
Fig. 4.1: *Pouring*	500k	128^3	0.724	0.071
Fig. 4.4: *Collision*	320k	96^3	0.542	0.061
Fig. 4.5: *Cylinder*	290k	96^3	0.497	0.055
Fig. 4.12: *Stirring*	250k	128^3	0.482	0.058
Fig. 4.14: *Multi-stage*	1100k	192^3	1.249	0.112
Fig. 4.18: *Multi-streams*	900k	192^3	1.045	0.098

Table 4.2. Statistics for the presented examples generated using our up-resing neural network *FFNet*. The discretization details (i.e., number of particles and velocity grid resolution) are shown for each example and the computation times per frame are expressed in seconds.

The training was performed for 300 epochs on 1296 pairs of simulations, taking an hour on average for each epoch computed on two nodes of four NVIDIA® GeForce® RTX 2080 Ti. For each pair of simulations, we used on average a window of 50 consecutive frames in order to capture most of the interactions of interest for learning. A longer temporal window may lead to a less accurate feature matching. We used a rather small batch size of 8 samples per iteration since our multi-resolution samples generate a significant memory footprint on the GPU. We used the ADAM optimizer [167] with an initial learning rate of 10^{-4} decaying at an exponential rate of 5% after each series of 50k iterations.

The *FFNet* architecture is composed of three downsampling convolution layers (detailed in Fig. 4.8), one embedding layer, and three upsampling convolution layers. An aggregation operation is performed after the first downsampling convolution layer to combine both input resolutions. Lastly, a linear flow regression layer is added at the end of the learning pipeline to output predicted particle displacements in $\mathbb{R}^3$. Table 4.4 in Appendix 4.9 shows the specifications of each MLP layer. We also use skip connections at each convolution level to concatenate the outputted local features between downsampling and upsampling layers. Finally, the MLP function h is activated by rectified linear units (ReLU) preceded by batch normalizations for both downsampling and upsampling convolution layers. The training and evaluation steps have been performed using the *TensorFlow* library. For performance reasons, the inference part has been implemented using the *TensorFlow* C++ API within the Autodesk *Bifröst* Graph®.

We chose a few baselines to compare the efficiency of our approach at the evaluation stage. In fairness and validity, we constrained the comparison to exclusively point-based learning methods. The evaluation metrics selected to compare the validity of these methods with respect to ours in the application context are the estimated position error (EPE) and the accuracy of the predicted displacement (Flow accuracy) when compared to ground truth. The EPE is evaluated as the average L_2 distance between the predicted and the ground-truth displacement vectors. Since the number of particles of the ground truth may differ from the up-resed one generated with our approach (i.e., upsampled $\boldsymbol{X}_l^*$ of the $\boldsymbol{X}_l$ coarse input), we use the closest particle to match each particle of the reference particle-based liquid. The flow accuracy measures the proportion of predicted displacements (using a small error margin $\epsilon = 0.001$ with respect to the scene scale) that are below a certain threshold (we used 0.1).

Method	**EPE**	**Flow accuracy**	**Convergence time**
PointNet [172]	0.45	26.11%	14.4
PointNet++ [173]	0.44	29.84%	13.6
FlowNet3D [169]	0.37	52.27%	10.3
FFNet (ours)	**0.23**	**63.71%**	**8.1**

Table 4.3. Flow estimation results on the *Colliding* dataset.

As highlighted in Table 4.3, our approach performs much better for the up-resing task on particle-based simulations when compared with the selected baselines. In addition to providing better results, our network converges much faster as shown in Fig. 4.10. The convergence times are expressed in hours per epoch. Moreover, we have noticed that with our approach, up to 40% of the EPE is due to static regions. These regions can show significant differences in volume when varying the simulation discretization parameters (i.e., particle spacing *ps* and grid resolution *gs*). When we manually exclude these regions (as suggested in § 4.7.2), the EPE reduces significantly.

Fig. 4.10 shows training losses evolving over iterations on several point-based learning methods. As shown, our approach offers a more steady error reduction over iterations as opposed to the presented baselines. In other words, our approach requires fewer iterations to

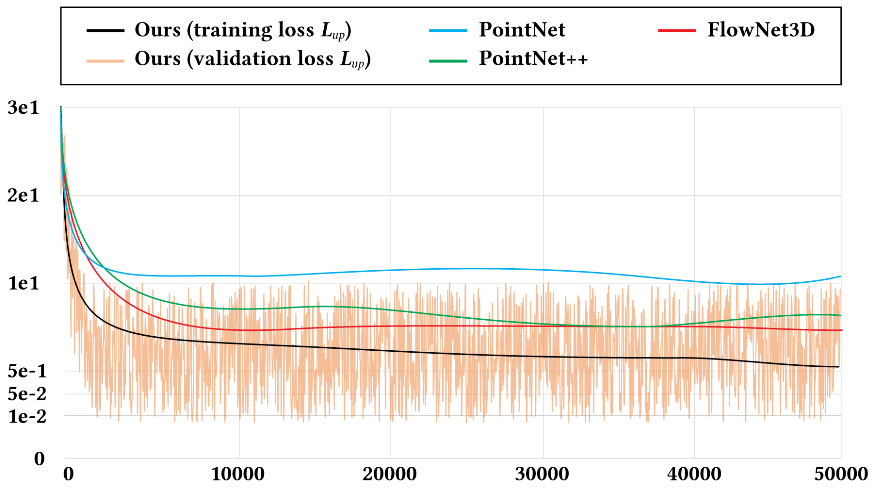


Figure 4.10. Comparing training losses on the *Container* dataset measured with our approach and a few point-based learning baselines. In order to lighten the chart, the validation loss (noisy orange curve) is showed only for our approach (black curve).

mimic the high-resolution details with good accuracy. Fig. 4.10 also highlights the generalization capabilities on a much more complex example (Fig. 4.18). As expected, we also have observed that similar examples from the training set take less time to converge to a decent level of detail (as compared to the corresponding reference). We observed a clear up-resing displacement after around 200 epochs as shown in Fig. 4.11. We noticed small differences in the precision of displacements in the range of 240 to 300 epochs (i.e., depending on the complexity of the input simulation), so for our purpose, training past that iteration threshold will unlikely bring significant value.

4.7.2. Evaluation and Discussion

Up-Resing Coarse Input. In the presented results, we provide as input to our network the particle positions and a displacement field obtained between the current frame and the next one. We use these displacements to act as deformation preconditioners to our network (i.e., hint on spatial and temporal deformations). More importantly, using these preconditioners as deformations prevents us from computing an actual deformation field $\boldsymbol{u}_{\text{up}}$ (i.e., using the *UpFlOF* algorithm) which would defy the objective of this approach by requiring a high-resolution input. The generated displacement by our network is then combined with the

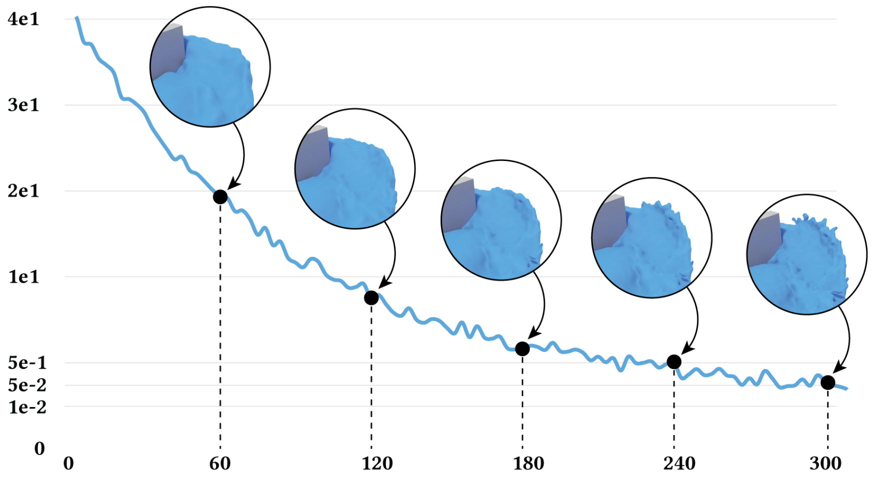


Figure 4.11. Convergence plot of our $\mathcal{L}_{\text{up}}$ loss for the *Stirring* example from Fig. 4.12. We show the loss error (Y-axis) evolving throughout the training epochs (X-axis).

velocity of the coarse input to account for the coarse motion of the liquid since displacements are computed locally and with respect to their neighborhood.

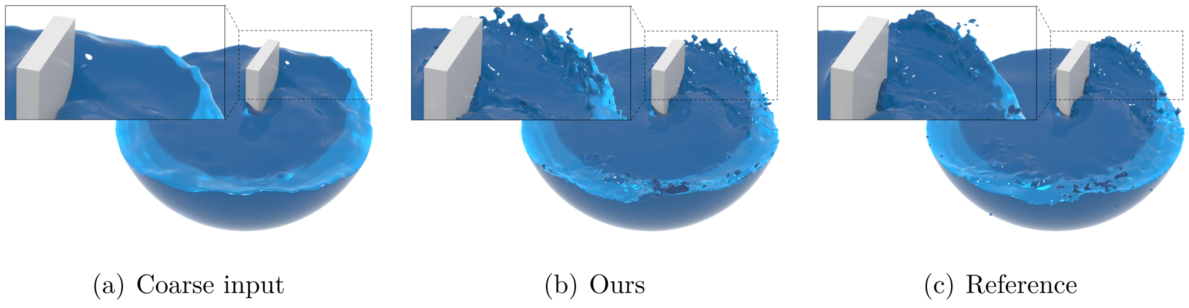


Figure 4.12. Comparison of our result (b) generated with *FFNet* using a low-resolution input (a) with the corresponding high-resolution reference (c).

For convenience, we initially decided whether to process selected regions or the whole volume of liquid. The reason is that generating up-resing details on static volumes of liquid introduces noise due to the regression operations on downsampled neighborhoods. Therefore, in some of our examples, we initially flagged regions (and contained particles) that might be of interest to provide compelling details often absent in coarse simulations. However, in some cases, it was simpler to predict displacements for the whole simulation domain and then smooth out selected regions of the generated surface.

Lastly, it was observed that droplets and highly diffuse volumes of liquid are not faithfully generated by our displacement network. One solution could be to flag diffuse particles as proposed by Um et al. [183]. For example, as exposed in the close-up insets of Fig. 4.12, our approach generates convincing splashing details while mostly remaining attached to the main volume of liquid. On the other hand, our network successfully reproduced eddies surrounding the moving obstacle that stirs the liquid. However, as compared with the high-resolution reference (Fig. 12(c)), we were unable to recreate the small droplets detached from the splashing portions of the liquid.

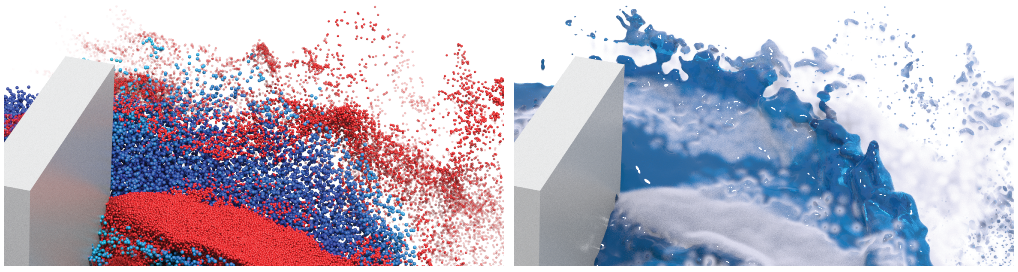

(a) Spray and foam particles (red) (b) Whitewater render

Figure 4.13. Spray and foam particles (red) added to our generated results (blue).

Nevertheless, these missing droplets can easily be procedurally added (as shown with the red particles in the inlet figure) on top of our displaced particles (e.g., using existing tools such as *Maya*® to create whitewater details). We have noticed that more static details such as smooth swirls are not fully reproduces by our approach. In these cases, the differences between low- and high-resolution liquids are subtle, and therefore, are underrepresented by our method of selecting samples whose window captures only specific liquid-liquid or solid-liquid behaviors (e.g., such as collisions, splashes, and so on). We provide an example to highlight this limitation in the supplemental video.

Fig. 4.1 is also a good example where adding spray and foam particles would improve the end results generated by our approach as it clearly lacks droplets compared to the high-resolution reference. On a related note, we have noticed that sparse neighborhoods require more iterations during training to converge and to reconstruct these small-scale details. In other words, these diffuse regions contribute less throughout iterations and would require more epochs during training with our network.

Generalization on More Complex Inputs. We have also tested the generalization capabilities of the proposed approach on more complex examples to evaluate how well it performs on unknown simulation setups. As exposed in § 4.7.1, our dataset is composed of solely single sources of disturbance either on static liquid or static obstacles. In Fig. 4.18, we compare our results with the ground truth on a three-streams example pouring into a single container. There are a few interesting features to observe in that figure. First, the small-scale details noticeable in the reference were successfully reproduced around the streams and at the impact points in the static liquid container. As previously mentioned, although our approach does not capture the diffuse particles very well (e.g., droplets), we can still observe a few detached chunks of liquid at the stream impacts.

Another interesting aspect observed in our results is the ability to reproduce the energy level of the simulations at higher resolutions. As noticeable in Fig. 4.18, the energy loss caused by a coarse discretization (Fig. 18(a)) is partially restored in appearance when upresed using our approach (Fig. 18(c)). The arced streamlines in our result (Fig. 18(c)) closely reproduce those of the high-resolution reference (Fig. 18(b)). It is also apparent in Fig. 4.14 that the static container is significantly more agitated in our result than it is in the coarse

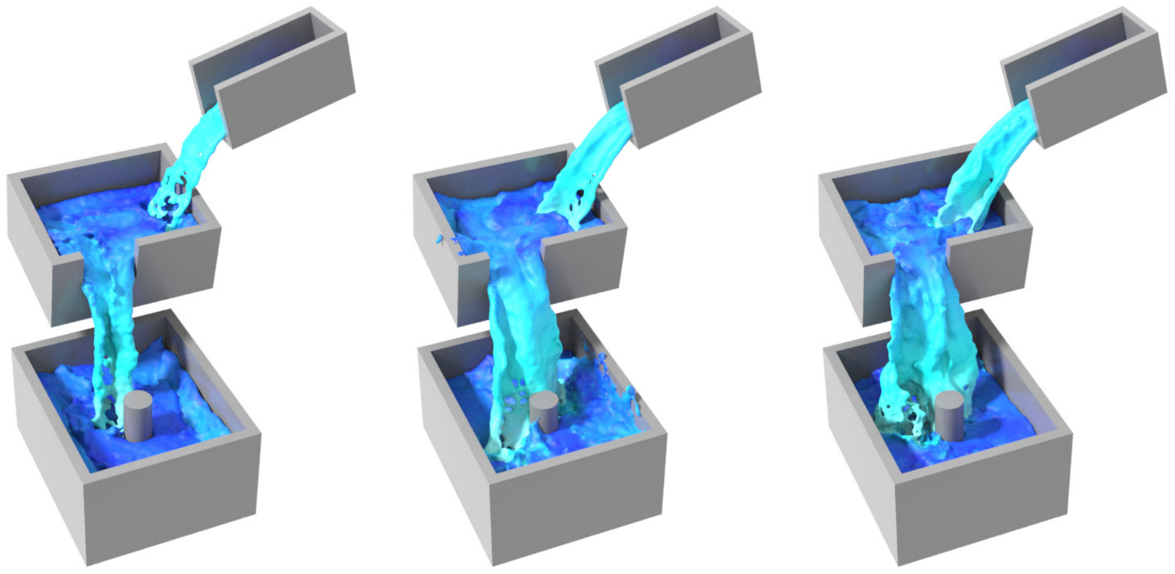

(a) Coarse input (b) Ours (c) Reference

Figure 4.14. Comparison of our result (b) generated with *FFNet* using a low-resolution input (a) with the corresponding high-resolution reference (c).

input simulation. At each level of that cascading stream, our approach was able to infer dynamic and turbulent behavior within impacted regions. In the lower container (i.e., the one with the cylinder obstacle), we have noticed that our result (Fig. 14(b)) was showing a slightly different but plausible turbulence behavior (Fig. 14(c)).

Improved Alignment. The alignment term (Eq. 4.4) is used to improve the precision of the deformation between resolutions. In order to fully reproduce high-resolution details on coarse liquids, the deformations applied with the inferred displacements of our network must be aligned in space and time to capture the small differences between the simulation resolutions. As an example, the impacts on the sphere shown in Fig. 4.1 present noticeable differences between the coarse and the high-resolution liquids. Our goal with that alignment in this particular example would be to ensure that we are able to capture the detailed splashing impacts of the reference and infer this on the coarser liquid. With the exception of the detached droplets, we show in Fig. 4.15 that our approach (top row in green) does a fairly good job at reproducing similar fine details around the impact crown as compared to the reference (bottom row in blue).

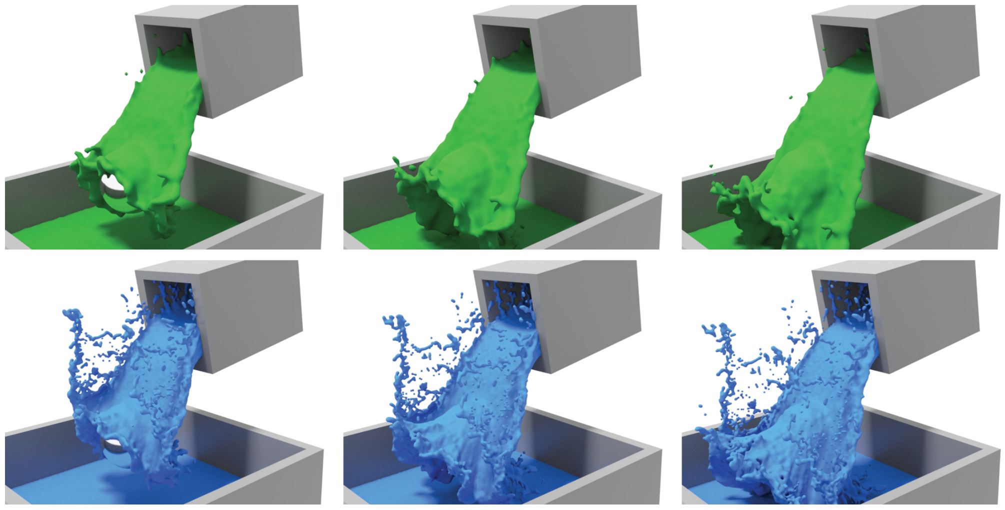


Figure 4.15. Comparing the evolution between our inferred liquid (top) and the high-resolution reference (bottom).

The benefit of using key-event alignment is illustrated in Fig. 4.16, where we compare the deformations applied with and without the alignment term **D**. We can observe that the green liquid fits better to the fine splashes of the reference (semi-transparent in blue) in comparison to the red liquid which presents the deformations before the correction on the

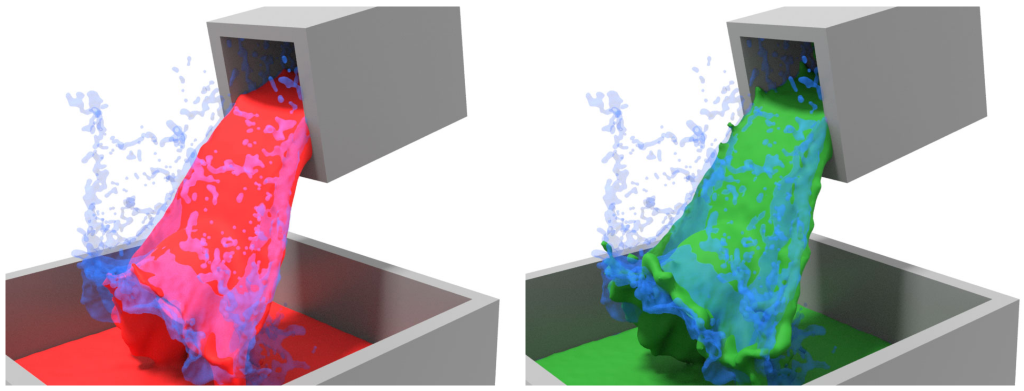
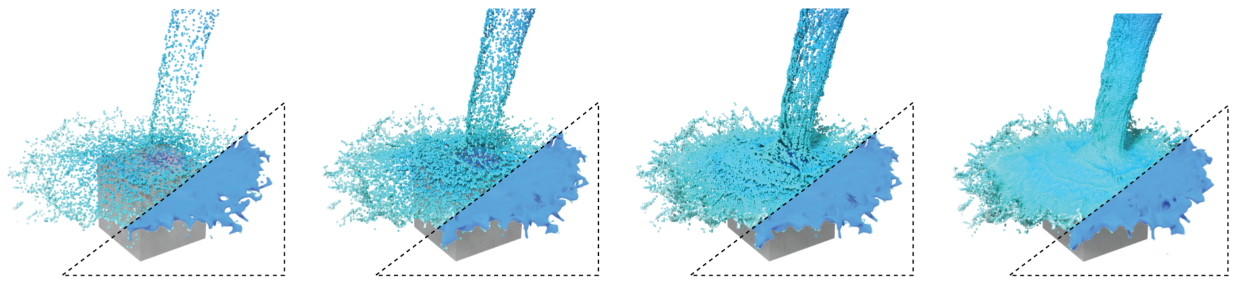

(a) Without key-event alignment (b) With **D**

Figure 4.16. Comparing without/with the proposed key-event alignment term on the example of Fig. 4.1.

alignment. The red liquid is missing most of the small-scale details of the main volume of liquid; this is due to a misalignment of the deformation field mapping the high-resolution to the low-resolution input.

Upsampling for Inference. The upsampling process of an input liquid prior to inference is a crucial step to fully appreciate the details generated by our approach. By analogy, this step is comparable to the need to have enough material to model fine details on a physical object. In our context, the material is discretized as particles. Applying our approach directly to a coarse liquid (i.e., without oversampling) would result in an extremely diffuse liquid. In fact, modeling so much detail through our inferred displacements only on a coarse liquid would simply spread the particles sparingly. In addition, an insufficient particle density may cause unwanted artefacts on the surface of the up-resed liquid. Furthermore, it is important to note that the input resolution at inference determines the displacement resolution of our network's output. In other words, the input resolution constrains the resolution of the inferred displacement. Nevertheless, the neighborhoods used by our convolution operators enable us to decouple our network from the input resolution while reproducing appealing details. Results shown in Figs. 4.14 and 4.18 are concrete examples of cases in which the upsampling of the input simulation made it possible to faithfully reproduce several apparent characteristics in the associated reference.

In Fig. 4.17, we show the influence of different distances d over the precision of the inferred displacements. As shown in that figure, the smaller the sampling radius, the higher



(a) $d = R$ (b) $d = 0.75R$ (c) $d = 0.5R$ (d) $d = 0.25R$

Figure 4.17. We show the influence of upsampling the particle band prior to the inference step. As shown in this figure, we qualitatively evaluated the results by varying the resampling radius of the narrow band.

the level of detail of the applied displacements. We have used $d = 0.5R$ (i.e., equal to the input particle separation radius) in most of the examples presented to limit the size of the input liquid to our network and because using a larger distance did not have a noticeable difference in the final result. Lastly, similarly to the work of Liu et al. [169], we also perform multiple passes of inference (usually 3-6 passes) using different upsampling distances of the surface band to reduce the inference noise and to prevent particles from emerging following the application of the displacements.

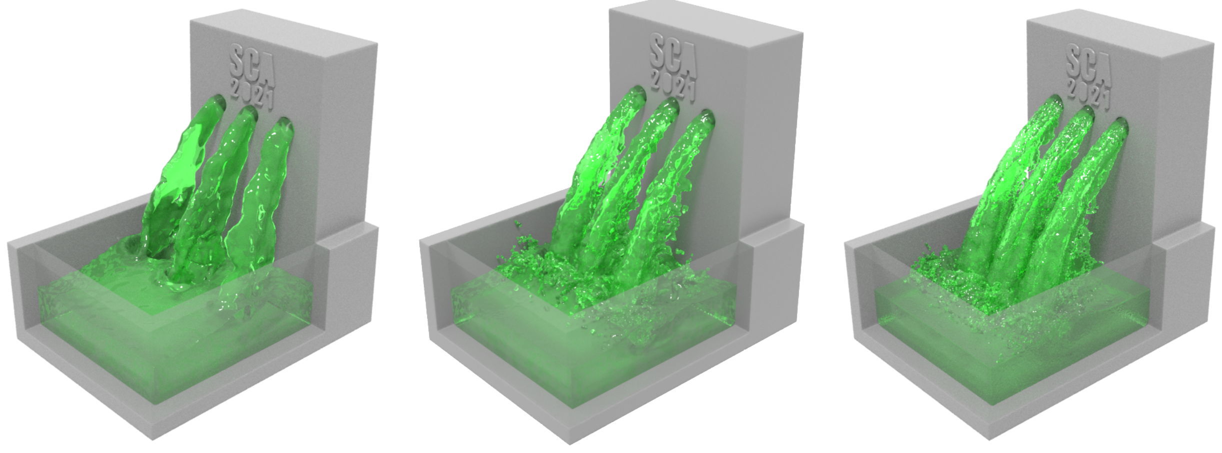
4.8. Conclusion and Future Work

We have presented an approach leveraging deep learning to increase the apparent resolution of a coarse particle-based liquid. In addition, we have proposed a framework using a state-of-the-art interpolation method to generate and augment a dataset of particle-based simulations for machine learning purpose. Our approach can infer plausible and complex details on the surface of low-resolution liquids, as illustrated in the examples in this paper and animated sequences in the accompanying video.

Looking toward the future, we believe that using a stacked network could improve our results with highly diffuse liquids and thin sheets. Our work could be combined with a spray particle classification network, such as proposed by Um et al. [183], to update an adaptive neighborhood within the convolution layers. Finally, we think that it would also be interesting to investigate volumetric learning methods targeting up-resing applications like this one but for smoke simulations.

Acknowledgments

This work was supported and funded by Mitacs Accelerate, Mitacs Globalink, NSERC, the Université de Montréal, and the École de technologie supérieure (ÉTS). Special thanks go to Nils Thuerey for his insightful advices, comments, and discussions during the first author's research internship at the Technical University of Munich (TUM) at the very beginning of this project. We thank Autodesk Inc. for providing resources to carry out the final stages of this project.



(a) Coarse input (b) Reference (c) Ours

Figure 4.18. Example highlighting the generalization capacities of our network in an unknown simulation setup.

References

[153] Mridul Aanjaneya, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Trans. on Graphics (TOG)*, 36(4), 2017.

[154] Kfir Aberman, Peizh Uo Li, Dani Lischinski, Olga Sorkine-Hornung, Daniel Cohen-Or, and Baoquan Chen. Skeleton-aware networks for deep motion retargeting. *ACM Trans. on Graphics (TOG)*, 39(4), 2020.

[155] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J Guibas. Adaptively sampled particle fluids. *ACM Trans. on Graphics (TOG)*, 26(3), 2007.

[156] Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Trans. on Visualization and Computer Graphics*, 18(8):1202–1214, 2012.

[157] Ryoichi Ando, Nils Thürey, and Chris Wojtan. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. on Graphics (TOG)*, 32(4), 2013.

[158] Markus Becker and Matthias Teschner. Weakly compressible SPH for free surface flows. In *Proc. Symposium on Computer Animation*, pages 209–217. ACM SIGGRAPH/Eurographics, 2007.

[159] Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. on Graphics (TOG)*, 36(4), 2017.

[160] Nuttapong Chentanez, Matthias Müller, and Tae-Yong Kim. Coupling 3D Eulerian, heightfield and particle methods for interactive simulation of large scale liquid phenomena. *IEEE Trans. on Visualization and Computer Graphics*, 21(10):1116–1128, 2015.

[161] Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with cnn-based feature descriptors. *ACM Trans. on Graphics (TOG)*, 36(4), 2017.

[162] Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A Efros, and Adrien Bousseau. 3D sketching using multi-view deep volumetric prediction. *Proc. ACM on Computer Graphics and Interactive Techniques*, 1(1):1–22, 2018.

[163] Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. Narrow band FLIP for liquid simulations. *Computer Graphics Forum*, 35(2):225–232, 2016.

[164] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Proc. International Conference on Neural Information Processing Systems*, pages 262–270, 2015.

[165] Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum*, 38(2):59–70, 2019.

[166] Theodore Kim, Jerry Tessendorf, and Nils Thuerey. Closest point turbulence for liquid surfaces. *ACM Trans. on Graphics (TOG)*, 32(2), 2013.

[167] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[168] L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Trans. on Graphics (TOG)*, 34(6), 2015.

[169] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3D point clouds. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 529–537, 2019.

[170] Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim, and Derek Nowrouzezahrai. Surface turbulence for particle-based liquid simulations. *ACM Trans. on Graphics (TOG)*, 34(6), 2015.

[171] Lukas Prantl, Boris Bonev, and Nils Thuerey. Generating liquid simulations with deformation-aware neural networks. In *International Conference on Learning Representations*, 2018.

[172] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 77–85, 2017.

[173] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. International Conference on Neural Information Processing Systems*, pages 5105–5114, 2017.

[174] Karthik Raveendran, Chris Wojtan, and Greg Turk. Hybrid smoothed particle hydrodynamics. In *Proc. Symposium on Computer Animation*, pages 33–42. ACM SIGGRAPH/Eurographics, 2011.

[175] Bruno Roy, Eric Paquette, and Pierre Poulin. Particle upsampling as a flexible post-processing approach to increase details in animations of splashing liquids. *Computers & Graphics*, 88:57–69, 2020.

[176] Bruno Roy and Pierre Poulin. A hybrid Eulerian-DFSPH scheme for efficient surface band liquid simulation. *Computers & Graphics*, 77:194–204, 2018.

[177] Bruno Roy, Pierre Poulin, and Eric Paquette. Neural upflow: A scene flow learning approach to increase the apparent resolution of particle-based liquids. In *Proc. Symposium on Computer Animation*. ACM SIGGRAPH/Eurographics, 2021.

[178] Takahiro Sato, Christopher Wojtan, Nils Thuerey, Takeo Igarashi, and Ryoichi Ando. Extended narrow band FLIP for liquid simulations. *Computer Graphics Forum*, 37(2):169–177, 2018.

[179] Barbara Solenthaler and Markus Gross. Two-scale particle simulation. *ACM Trans. on Graphics (TOG)*, 30(4), 2011.

[180] Nils Thuerey. Interpolations of smoke and liquid simulations. *ACM Trans. on Graphics (TOG)*, 36(1), 2017.

[181] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating Eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pages 3424–3433. PMLR, 2017.

[182] Hsiao-Yu Fish Tung, Hsiao-Wei Tung, Ersin Yumer, and Katerina Fragkiadaki. Self-supervised learning of motion capture. In *Proc. International Conference on Neural Information Processing Systems*, pages 5242–5252, 2017.

[183] Kiwon Um, Xiangyu Hu, and Nils Thuerey. Liquid splash modeling with neural networks. *Computer Graphics Forum*, 37(8):171–182, 2018.

[184] Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2019.

[185] Gokul Varadhan, Shankar Krishnan, TVN Sriram, and Dinesh Manocha. Topology preserving surface extraction using adaptive subdivision. In *Proc. Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 235–244, 2004.

[186] Tuanfeng Y Wang, Hao Su, Qixing Huang, Jingwei Huang, Leonidas J Guibas, and Niloy J Mitra. Unsupervised texture transfer from images to model collections. *ACM Trans. on Graphics (TOG)*, 35(6), 2016.

[187] Zirui Wang, Shuda Li, Henry Howard-Jenkins, Victor Prisacariu, and Min Chen. Flownet3d++: Geometric losses for deep scene flow estimation. In *Proc. IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 91–98, 2020.

[188] Maximilian Werhahn, You Xie, Mengyu Chu, and Nils Thuerey. A multi-pass GAN for fluid flow super-resolution. *Proc. ACM on Computer Graphics and Interactive Techniques*, 2(2):1–21, 2019.

[189] Steffen Wiewel, Moritz Becher, and Nils Thuerey. Latent space physics: Towards learning the temporal evolution of fluid flow. *Computer Graphics Forum*, 38(2):71–82, 2019.

[190] Rene Winchenbach, Hendrik Hochstetter, and Andreas Kolb. Infinite continuous adaptivity for incompressible SPH. *ACM Trans. on Graphics (TOG)*, 36(4), 2017.

[191] Rene Winchenbach and Andreas Kolb. Optimized refinement for spatially adaptive SPH. *ACM Trans. on Graphics (TOG)*, 40(1), 2021.

[192] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Deforming meshes that split and merge. In *ACM SIGGRAPH 2009 papers*, pages 1–10. 2009.

[193] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Trans. on Graphics (TOG)*, 37(4), 2018.

[194] Cheng Yang, Xubo Yang, and Xiangyun Xiao. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016.

[195] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. on Graphics (TOG)*, 24(3):965–972, 2005.

4.9. Appendix A: Network Architecture Layer Specifications

We provide more details on the specifications of the layers composing our *FFNet* network. The neighborhood radius R is expressed as a multiplication factor of the input particle separation ps. On the other hand, the sample rate is expressed as a proportion of the sample points kept from one layer to another. These proportions were obtained during several experiments aiming for the best visual-performance tradeoff at evaluation. The $N \times 3$ displacement vectors are the result of the last linear regression layer. Lastly, we expose the number of nodes as *width* for each layer.

Layer	R	**Sample rate**	**Width**
Down. Conv.	$2 \cdot ps$	0.5×	{32,32,64}
Down. Conv. + Agr.	$4 \cdot ps$	0.25×	{64+64,64+64,128+128}
Down. Conv.	$8 \cdot ps$	0.125×	{256,256,512}
Embedding	$8 \cdot ps$	–	{256,256,512}
Up. Conv.	$8 \cdot ps$	2×	{128,128,256}
Up. Conv.	$4 \cdot ps$	4×	{128,128,256}
Up. Conv.	$2 \cdot ps$	2×	{128,128,128}
Linear	–	–	3

Table 4.4. Specifications of the MLP layers used in our network architecture.

4.10. Appendix B: Evaluation Times Breakdown

The evaluation steps were performed on an NVIDIA RTX® A6000 with 48 GB of GDDR6 memory. For each example in Table 4.2, we present a breakdown of the computation times (in seconds) under *FFNet eval.* as shown in Table 4.5.

Example	**Reference**	***FFNet* evaluation**			
		Particle Upsampling	Network Prediction	*UpFlOF* Advection	**Speed-up**
Fig. 4.1	6.9	0.012	0.043	0.016	16×
Fig. 4.4	5.2	0.010	0.038	0.013	15×
Fig. 4.5	4.9	0.009	0.034	0.012	13×
Fig. 4.12	4.8	0.009	0.036	0.013	17×
Fig. 4.14	11.1	0.020	0.067	0.025	19×
Fig. 4.18	10.3	0.017	0.059	0.023	17×

Table 4.5. Breakdown of the evaluation times to generate our results using the prediction displacements of our network *FFNet*. Note that we include the upsampling and advection steps in the evaluation times.

The *FFNet* evaluation step includes the particle upsampling, the displacement predictions by our network, and the advection using the *UpFlOF* advection scheme. Each step has been implemented using CUDA capabilities to offer interactive computation times. Both upsampling and advection steps are really fast to compute. Predicting displacements takes most of the *FFNet* evaluation time since this step requires reading the pre-trained model. The reference column shows the computation times per iteration of the high-resolution ground truth (composed of 5M to 20M particles). Our approach provides speed-ups between 13× and 19× faster than the references.

4.11. Appendix C: Multi-resolution Datasets

Each simulation sample used to generate our datasets is composed of a pair of liquids (each pair using the same initial conditions): a coarse low resolution and a detailed high resolution. In Fig. 4.19, we present a small subset of our datasets.

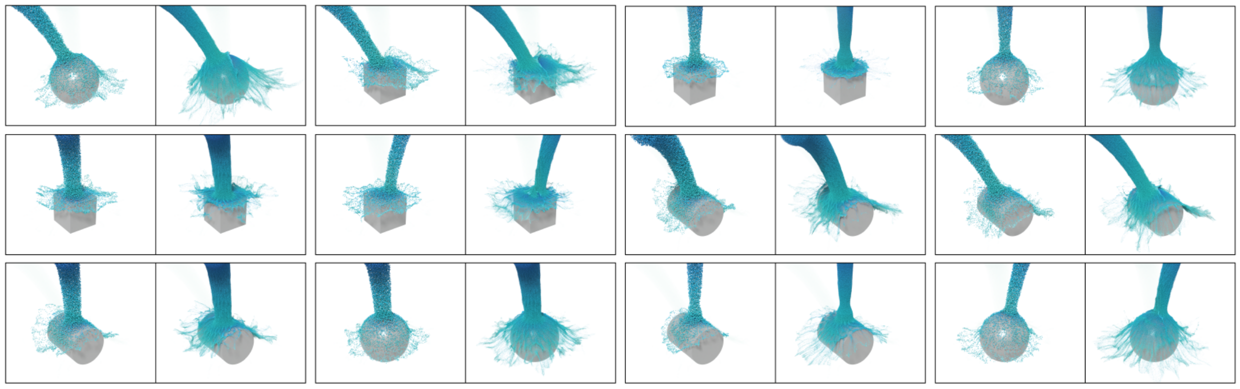


Figure 4.19. A small subset of simulation pairs from our multi-resolution *Colliding* dataset used to train the proposed *FFNet* model.

4.12. Appendix D: Notation

Table 4.6 details our notation. Generally speaking, matrices are in bold uppercases, vectors in bold lowercases, and single values in italics. A dotted field is actually for one particle of that field.

Symbol	**Description**	**Symbol**	**Description**
$\boldsymbol{\Phi}$	Signed Distance Function	$\boldsymbol{X}_l$	Low-resolution particle set
$\boldsymbol{u}$	Deformation field (Thuerey's)	$\boldsymbol{X}_h$	High-resolution particle set
$\boldsymbol{u}_\omega$	Predicted velocity field	$\boldsymbol{X}'$	Upsampled particle set
$\dot{\boldsymbol{u}}$	Velocity of a particle	f	Local feature generated
$\boldsymbol{u}_{\text{up}}$	Deformation field for up-resing	f^*	Weighted local feature
E_d	SDF energy term	n_j	Neighborhood j
β_S	Smoothness coefficient	p_i	Particle i
E_{smooth}	Smoothness regularizer	w	Weight
β_T	Tikhonov coefficient	k	Kernel function
E_{Tikhonov}	Tikhonov regularizer	$\mathcal{L}_{\text{up}}$	Loss function
$\boldsymbol{A}_{\text{UpOF}}$	Discretized energy term	$\boldsymbol{\omega}$	Predicted displacement
$d_{\boldsymbol{x}_i \rightarrow \boldsymbol{x}_k}$	4D Euclidean proportional coefficient	$\boldsymbol{\omega}^*$	Ground truth displacement
$\boldsymbol{x}_k$	Closest key surface point	$\dot{\boldsymbol{\omega}}$	Displacement of a particle
$\boldsymbol{x}_i$	Surface point	λ_{n_j}	Adaptive weight
$\bar{\boldsymbol{x}}$	Weighted average of the coordinates	d_b	depth of narrow band
P	Pair of multi-resolution simulations	d_{MAC}	Extrapolation distance
$\boldsymbol{\Theta}$	Matrix of initial conditions	α_{FlOF}	Interpolation weight

Table 4.6. Notation and associated meaning for our terms.

Chapter 5

Conclusion

In this thesis, we presented three papers in which we propose approaches to improve the efficiency, parameterization, and quality of particle-based liquid simulations. From our contributions, we were able to reduce the dependencies of the SPH method to particles [201], decouple realistic splashing details from the simulation model [200], and infer high-resolution features on coarse particle-based liquids [202]. These contributions were made with the common goal of improving current methods leading to efficient, configurable, and interactive high-resolution liquid simulations for the media and entertainment community. When combined, these contributions can pave the way for significant improvements over currently available methods. Foremost, as the tracer particles already benefit from artificial pressure forces similar to SPH methods, integrating our splash model into the proposed hybrid Eulerian-DFSPH would make sense. In addition, we could easily use this combined approach to generate hybrid SPH liquid animations for our up-res neural approach. On top of learning displacements, we could stack another network, i.e., before *FFNet*, to classify diffuse particles and treat them as ballistic as we do in our second contribution [200].

In the following sections, we discuss our published contributions along with future research avenues that we believe are worth exploring. We conclude by opening on thoughts related to untapped and promising research topics.

5.1. Redefining Hybrid Models for Fluids

The first paper [201] provides a hybrid scheme to simulate a narrow band of SPH particles coupled with an Eulerian grid for in-depth and coarser flows. Our approach leverages adaptivity by focusing the details using particles exclusively around the surface and complex dynamic boundaries.

Continuous Adaptivity. As mentioned in the paper, our approach suffers from instability in small volumes of liquid where the number of SPH particles is insufficient with respect to the number of fictitious particles. One way to resolve this is by treating these small volumes using solely the SPH simulation part of the model (i.e., without the Eulerian grid and the fictitious particles). Another interesting and more efficient way to resolve this issue would be to use an adaptive variant of the fictitious particles, as recently proposed by Winchenbach et al. [204, 205].

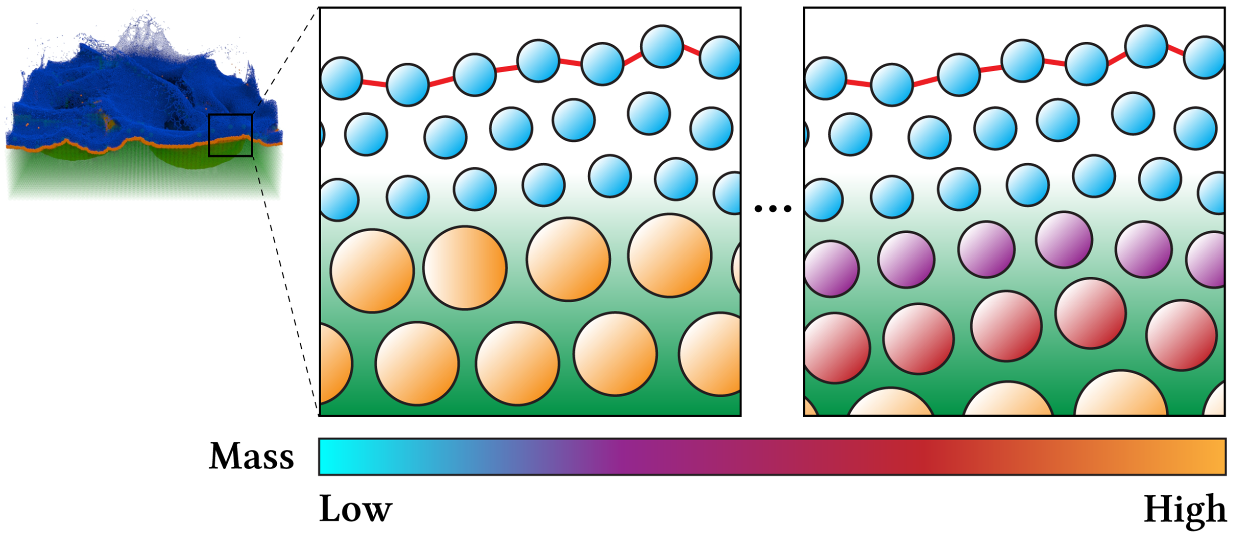


Figure 5.1. Comparing our current particle band model (left enlarged view) with a continous adaptive band model (right enlarged view).

As shown in Fig. 5.1, our current particle band model (left enlarged view) has significant mass discontinuities between the SPH and fictitious particles, inevitably causing energy dissipation during transfers of the hybrid steps. By using continuous particle sizes and thus allowing a smoother transition in masses at the boundaries between the two models (right enlarged view), our narrow band of particles could handle thin sheets of liquid while remaining stable. The challenging part would remain to define a precise way to threshold the transition between the continuous adaptive band of particles and the Eulerian grid. This threshold mass could be expressed as the contribution accuracy of the underlying grid cells.

Unbounded Simulation Domain. On a related note, although hybrid methods have provided significant advances to take advantage of particles while remaining stable and efficient, relying on a grid-based simulation below the liquid surface inevitably restricts the domain of simulation. Even if their work has been solely introduced to Eulerian methods, we believe that combining Kelvin transformations as proposed by Nabizadeh et al. [191] would overcome the limitations related to bounded simulation domains. Naturally, coupling the particles with that infinite domain could be challenging with the boundaries and highlighted singularities within the resulting domain.

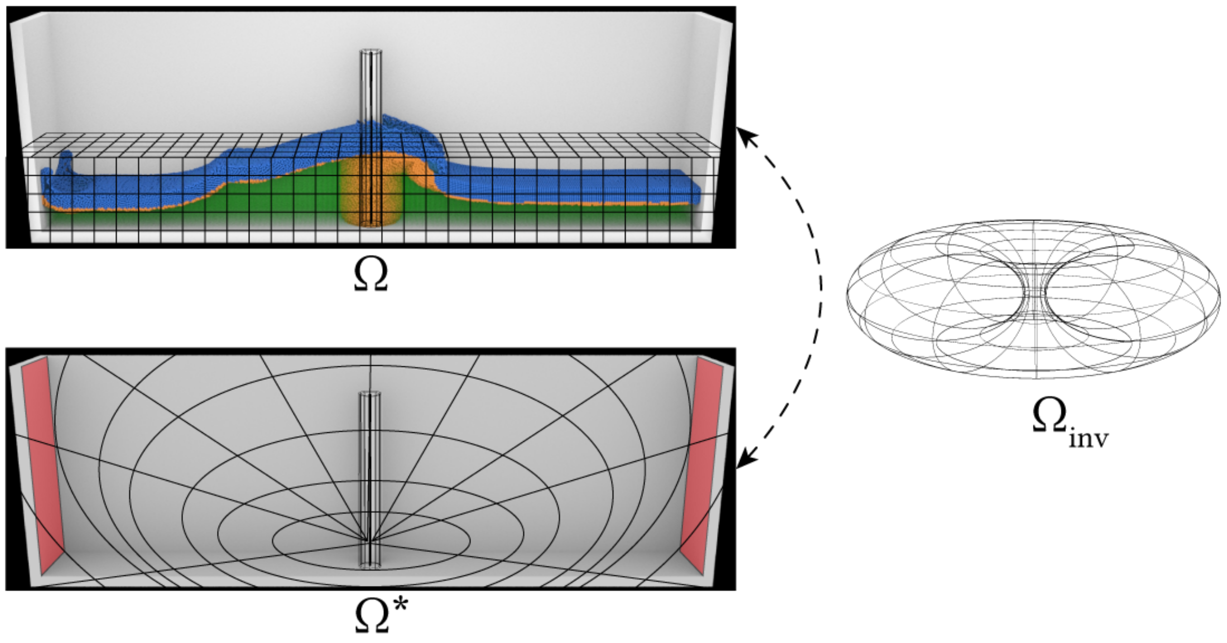


Figure 5.2. Infinite simulation domain generated from Kelvin Transform.

Using the Kelvin Transform to invert the domain (i.e., Ω_{inv}) distances from the origin of Ω, we could extend our current Eulerian domain to an infinite horizon as defined by Ω^*. We can imagine simulating a dam break scenario based on the scene shown in Fig. 5.2. In such a scenario, the walls (in red of the bottom image) could be shattered, letting the liquid expend itself past the initial container. Once the infinite domain is generated, the band of particles can be coupled on top of Ω^*, as introduced with our approach.

5.2. Decoupling Splash Dynamic

Although SPH is considered more physically accurate, FLIP remains faster, more stable, and opens up to many possible workflows that are not possible with the SPH model. Our second paper [200] benefits from the FLIP model as we introduce a novel way to integrate diffuse particles in splashing liquids. In our paper, we present a post-processing approach to simulate splashing details on top of existing liquid animations. With nearly no additional cost, our volume-based method allows us to interactively generate multiple variants of the same splashing liquid. Moreover, our implicit wave model introduces a seamless and efficient way to couple procedurally added particles with a coarse volume of liquid for improved interactions.

Improved Splash Dynamics. Nevertheless, as highlighted in our paper, pre-computing the splash volume requires going through each frame of the input animation. This process is computationally expensive even though it needs to be computed only once for a specific scenario. Given the availability of the simulation data, such a pre-computed volume could be adaptively predicted from a small temporal window (i.e., 4D data). As opposed to combining volumes throughout the animation, we could compute a smaller intermediate volume generated from detecting possible diffuse particles. Recently, the work of Um et al. [203] showed that detecting diffuse particles could generate convincing splashes. Inspired by their work, we could train a classifier to predict the likelihood of particles being diffused. In contrast to their proposed splash modeling, we would keep the upsampling and the ballistic steps to preserve the controllability of the approach.

Stylization through Diffuse Material. Another interesting research avenue would be to reformulate diffuse material (i.e., foam, spray, and droplets) as a spatio-temporal *style* for liquids. Similar to Liu and Jacobson [197], we could define the dynamic differences between a low- and a high-resolution liquid as differences between the associated two generated surfaces for a single frame of animation.

By optimizing that expression as a loss term with respect to the ground truth, it would be possible to learn the diffuse behavior occurring at high resolution as a *style* that can be applied to a coarse liquid surface. As shown in Fig. 5.3, the surface of the input coarse liquid would be considered as the base to which we add the dissipated details visible in the high-resolution reference.

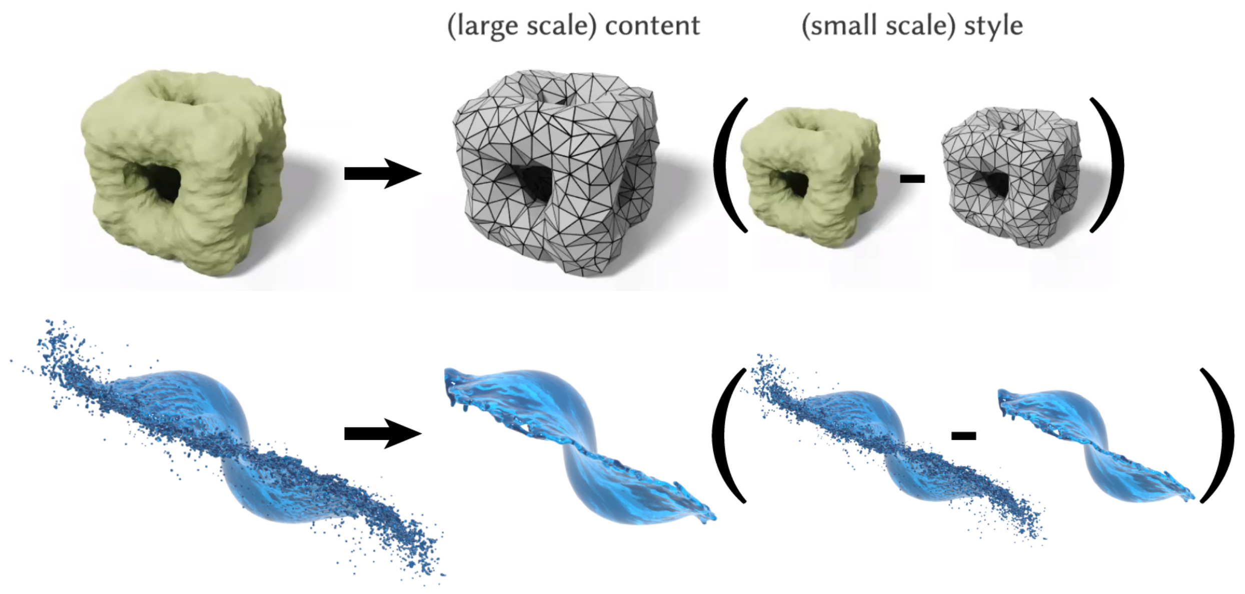


Figure 5.3. Applying style learning [197] (top) to generate a detailed liquid surface from a coarse base surface (bottom).

5.3. Towards Resolution-Agnostic Models

Our third paper [202] presents an up-resing approach for particle-based liquids using a deep neural network architecture. By combining particle neighborhoods to encode similarities between discretizations with our deformation prior, we provide consistent up-resing details throughout a broad range of simulation settings. In addition, we decouple the inference part from the advection part to offer a more robust generalization over a broad spectrum of simulation scenarios.

Pure Lagrangian Displacement. For future work, as we initially process the high-resolution features as position displacements, using position-based dynamics [199] would make sense with our approach as our goal is to offer a purely Lagrangian approach for particle-based learning on liquids. Even though our approach currently benefits from a velocity grid for pre-computed deformations, skipping the semi-Lagrangian advection part could improve its scalability to other types of particle-based liquids. Furthermore, it would be interesting to look into ways of proposing a *simulation-driven* approach using a position-based fluid [198]. Such a position-based approach could allow the integration of position constraints directly into the interleaved network, enforcing both constant density and incompressibility.

Discretization-Free Models. The discretization of physical phenomena such as liquids can introduce various artifacts, including dissipated small-scale details. As we represent a liquid with more elements (i.e., particles, grid cells, etc.), we theoretically tend towards the exact solution. However, getting the exact solution in a discretized context means that we are using an infinity of elements to simulate our physical phenomenon, which is impossible for numerical methods. Nevertheless, there is an actual trade-off between the number of elements used and the visual quality. At a certain scale along with the discretization, pushing it further will only slightly change the generated solution, making it unnecessary to get past a certain threshold. Even when using an analytical solution, numerical methods are required to be evaluated for a finite number of values over a finite number of times.

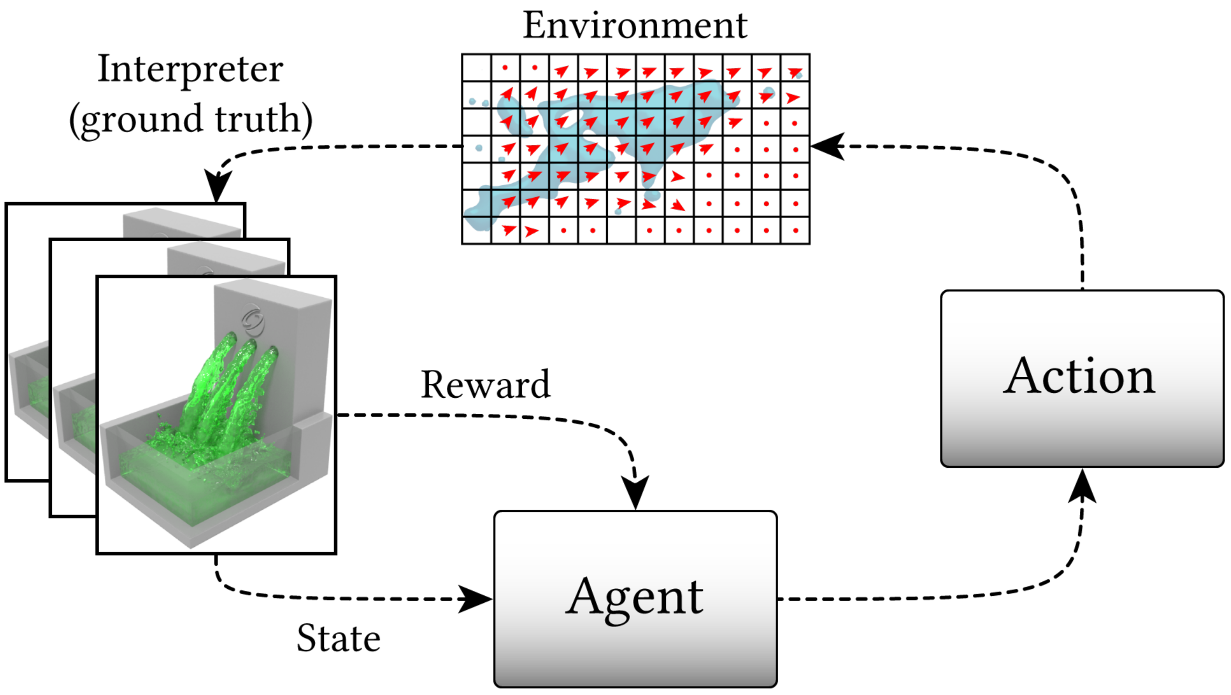


Figure 5.4. Reinforcement learning pipeline to approximate the simulation discretization.

That being said, learning a quasi-optimal way (i.e., as close as possible to the ground truth) to discretize an animated liquid could partially alleviate the painful process of discretization toward an adequate solution. As previously introduced to better approximate PDEs [196], we could optimize a neural network to learn to discretize a volume of fluid with a suitable representation (e.g., using an adaptive grid). Such a learning pipeline could be trained using real examples captured as videos to encapsulate both the spatial and temporal features. A reinforcement learning method could be effective to learn such features (Fig. 5.4). An agent could approximate a liquid by applying a distinct scale of discretization as the action on a specific scenario using pre-determined initial conditions. The state of the liquid would be evaluated over time until the generated simulation reaches a certain level of precision when compared to the ground truth (e.g., could be from a reference video). Following that evaluation (i.e., based on policies), the action on the environment used to update the solution is penalized or rewarded depending on the outcome (i.e., getting closer to the solution or not).

5.4. Above and Beyond

We believe that several untapped research tracks on fluid simulations are worth exploring in more depth. We take a step back here and make a few observations based on our experience accumulated throughout the presented contributions.

As we advanced through this journey of modeling simulation models, numerical methods to approximate the complex behavior of fluids have slowly morphed from analytic models to robust and precise iterative models. But even today, modeling phenomena such as fluids in a discretized manner remains challenging. Although iterative models have proven to be efficient to reproduce fluid dynamics, the way computers force us to use creativity to discretize these phenomena is just the tip of the iceberg. As an example, over the last two decades, several methods were proposed to use adaptivity to represent fluid elements in a more flexible way to encapsulate the Navier-Stokes equations. However, with most of these methods, an element of fluid has been defined as a particle or a grid cell, or many variants of them. Nevertheless, there might be more suitable structures that could be used to simulate fluids. Multi-scale methods are valid examples of this ongoing tendency. Similar to generative design, there might be more suitable and unintuitive organic structures that would better fit this purpose. Deep learning also offers promising avenues towards this goal by providing efficient ways to extract meaningful and unobservable features of fluids from latent space. On a related note, advection of implicit surfaces and meshes is a promising avenue to properly define the evolution of fluid volumes.

Finally, between the hybrid and unified methods lies an inherent challenge that is the way we observe the motion of these fluid elements. Using time integrators, we are capable of looking at the current and previous states to approximate a future state. Given the amount of data available and keeping in mind that machine learning techniques are getting better to predict future states based on examples and experiences (e.g., GAN), we can imagine that time integrators currently used in state-of-the-art approaches could benefit from these. On a different note, there is also the idea of transfer between Eulerian and Lagrangian approaches. Nonlinear models could be used through deep learning to express bijective mapping functions between different viewpoints. Again, a remaining challenge would be to define a meaningful function connecting low-resolution to high-resolution elements for staying resolution agnostic. By using an operator projecting the Eulerian elements in Lagrangian space and vice versa, we would be able to adequately assess the efficiency of the inference with respect to ground truth.

References

[196] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretizations for partial differential equations. *Proc. of the National Academy of Sciences*, 116(31):15344–15349, 2019.

[197] Hsueh-Ti Derek Liu and Alec Jacobson. Cubic stylization. *ACM Trans. on Graphics (TOG)*, 38(6):1–10, 2019.

[198] Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. on Graphics (TOG)*, 32(4):1–12, 2013.

[199] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[200] Bruno Roy, Eric Paquette, and Pierre Poulin. Particle upsampling as a flexible post-processing approach to increase details in animations of splashing liquids. *Computers & Graphics*, 88:57–69, 2020.

[201] Bruno Roy and Pierre Poulin. A hybrid Eulerian-DFSPH scheme for efficient surface band liquid simulation. *Computers & Graphics*, 77:194–204, 2018.

[202] Bruno Roy, Pierre Poulin, and Eric Paquette. Neural upflow: A scene flow learning approach to increase the apparent resolution of particle-based liquids. In *Proc. Symposium on Computer Animation.* ACM SIGGRAPH/Eurographics, 2021.

[203] Kiwon Um, Xiangyu Hu, and Nils Thuerey. Liquid splash modeling with neural networks. *Computer Graphics Forum*, 37(8):171–182, 2018.

[204] Rene Winchenbach, Hendrik Hochstetter, and Andreas Kolb. Infinite continuous adaptivity for incompressible SPH. *ACM Trans. on Graphics (TOG)*, 36(4):1–10, 2017.

[205] Rene Winchenbach and Andreas Kolb. Optimized refinement for spatially adaptive SPH. *ACM Trans. on Graphics (TOG)*, 40(1):1–15, 2021.