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

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

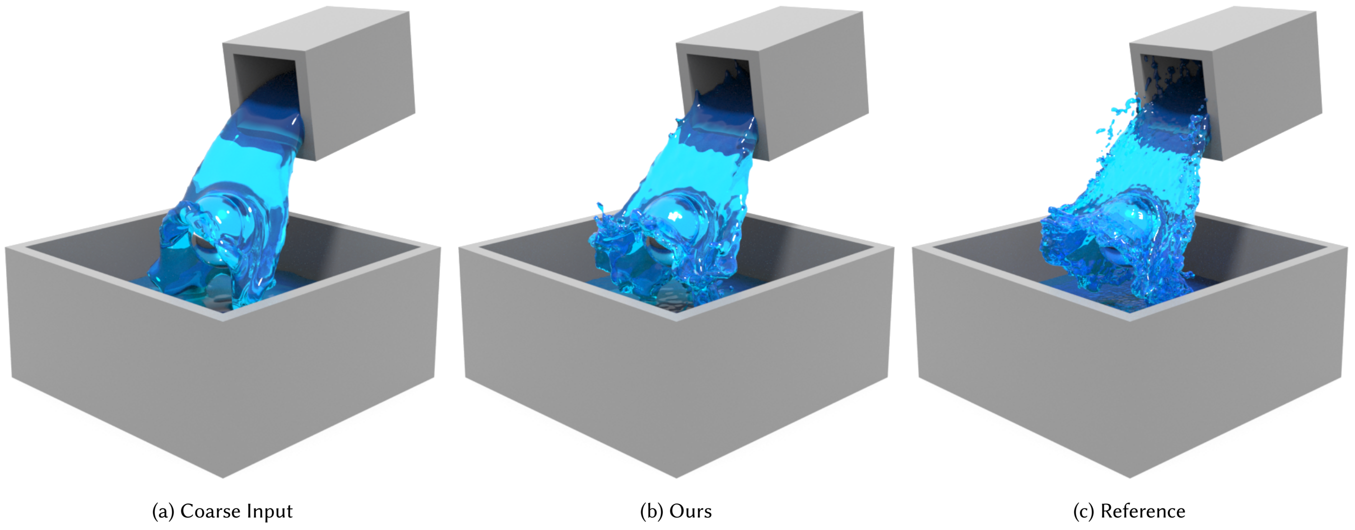


Fig. 1. Our learning deformation field reproduced (b) most of the small- and large-scale details from the high-resolution ground truth (c) using solely the low-resolution input example (a). Particles are added and displaced by our network to generate *plausible* details of up-resed animations at a fraction of a high-resolution simulation cost.

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.

CCS Concepts: • **Computing methodologies → Physical simulation**.

Additional Key Words and Phrases: fluid simulation, particle-based liquid, deformation field, optical flow, up-resing, machine learning, deep neural network

**ACM Reference Format:**
Bruno Roy, Pierre Poulin, and Eric Paquette. 2021. Neural UpFlow: A Scene Flow Learning Approach to Increase the Apparent Resolution of Particle-Based Liquids. *ACM Trans. Graph.* 1, 1, Article Submission to SCA'21: # 31 (August 2021), 16 pages. https://doi.org/0000001.0000001_2

Authors' addresses: Bruno Roy, Université de Montréal, Canada and Autodesk, Canada, bruno.o.roy@umontreal.ca; Pierre Poulin, Université de Montréal, Canada, poulin@iro.umontreal.ca; Eric Paquette, École de technologie supérieure, Canada, eric.paquette@etsmtl.ca.

## 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 [Aberman et al. 2020], motion capture [Tung et al. 2017], style transfer [Wang et al. 2016], 2D-to-3D sketching techniques [Delanoy et al. 2018], physically-based rendering [Chaitanya et al. 2017], texture synthesis [Gatys et al. 2015], fluid simulations [Kim et al. 2019], 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 [Ladický et al. 2015; Tompson et al. 2017; Yang et al. 2016], to compute fluid features [Kim et al. 2019; Um et al. 2018; Ummenhofer et al. 2019; Wiewel et al. 2019], and to infer visual details [Chu and Thuerey 2017; Prantl et al. 2018]. 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 [Qi et al. 2017a,b] 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 [Liu et al. 2019; Wang et al. 2020] 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. [2018]. 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 [2017] 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.

## 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 [Adams et al. 2007; Winchenbach et al. 2017]. While these methods share the same goal in spirit, Winchenbach et al. [2017] 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 [2011] introduced an adaptive method to couple a dual particle-layer scheme using two distinct particle resolutions. The method of Winchenbach et al. [2017] was also recently extended to improve spatial adaptivity by introducing a continuous objective function as a refinement scheme for SPH [Winchenbach and Kolb 2021].

A method focusing on preserving thin sheets was introduced by Ando et al. [2012] 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 [Ferstl et al. 2016]. The method was first introduced to *FLIP* given its semi-Lagrangian nature. The method was then extended by Sato et al. [2018] 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 [Chentanez et al. 2015; Raveendran et al. 2011; Roy and Poulin 2018]. Other adaptive methods were also proposed to exploit spatially adaptive structures to capture different simulation scales [Aanjaneya et al. 2017; Ando et al. 2013].

Meanwhile, procedural methods have also been introduced as a refinement scheme by generating surface points through wave simulations. As noted by Kim et al. [2013], 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. [2013], which was applied in an Eulerian setup (i.e., solely based on the corresponding SDF and the velocity field), Mercier et al. [2015] 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 [Roy et al. 2020].

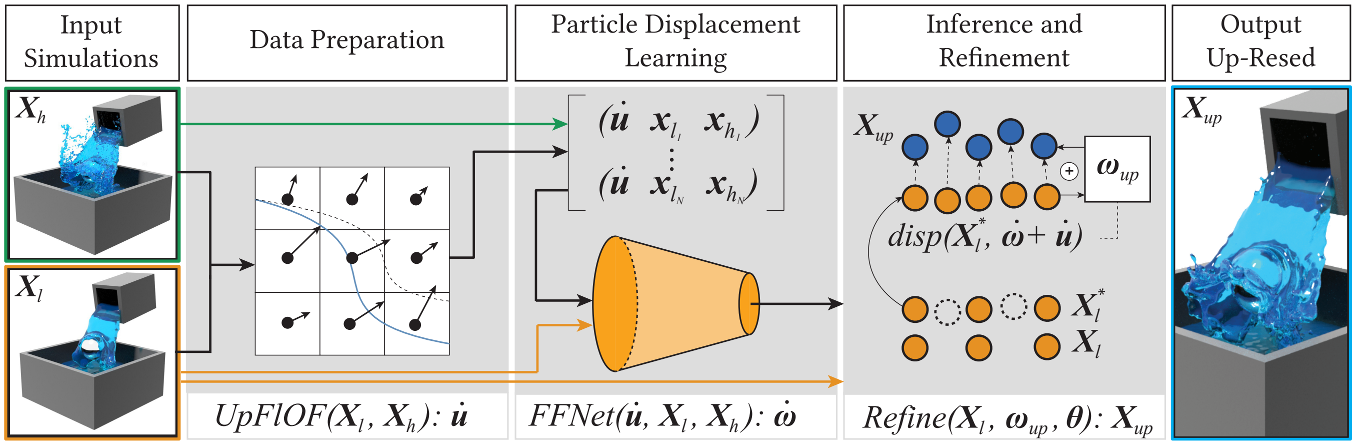

Input Simulations
Data Preparation
Particle Displacement Learning
Inference and Refinement
Output Up-Resed
$X_h$
$X_l$
$(\dot{u}\ x_{l_1}\ x_{h_1})$
$(\dot{u}\ x_{l_N}\ x_{h_N})$
$X_{up}$
$\omega_{up}$
$disp(X_l^*, \dot{\omega} + \dot{u})$
$X_l^*$
$X_l$
$X_{up}$
$UpFlOF(X_l, X_h): \dot{u}$
$FFNet(\dot{u},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

Fig. 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).

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. [2015]. 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 [Yang et al. 2016]. More recently, NNs were combined with numerical solvers to model diffuse particle statistics for the generation of splashes [Um et al. 2018]. Convolutional neural networks (CNN) were also widely used, but instead to predict Eulerian simulation properties. Thompson et al. [2017] 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 [Chu and Thuerey 2017]. 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 [Xie et al. 2018]. Meanwhile, Kim et al. [2019] 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 [Werhahn et al. 2019]. Due to their sequential properties, LSTM-based layers were later used by Wiewel et al. [2019] 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. [2019]. 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. [2018] 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. [2018], 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.

## 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* [Liu et al. 2019]; it encodes displacements between two unstructured and unordered particle-based liquids simulated at different resolutions using solely their positions and velocities as input (§ 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 (§ 4.2). Similarly to the work of Thuerey [2017], 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 (§ 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. 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. 2, we use dotted symbols for the Lagrangian properties. For instance, the velocity of a particle would be noted as $\dot{\boldsymbol{u}}$.

***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 INTER-RESOLUTION LIQUID INTERPOLATION

In this section, we adapt an algorithm originally from Thuerey [2017] 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. 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. 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.

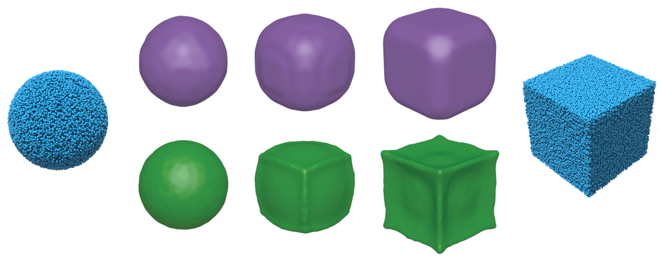


Fig. 3. A simple case where a sphere is morphed at three different steps (b) into a cube: comparing our interpolation approach applied directly on particles (bottom in green) with a state-of-the-art interpolation method [Thuerey 2017] applied on the corresponding SDF (top in purple).

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 [2017], we use a 4D optical solve to interpolate between input scenarios using the same initial conditions, but at different resolutions (low and high resolutions).

### 4.1 Up-Resing Optical Solve

***Inter-Resolution Optical Flow.*** We employ an approach similar to the work of Thuerey [2017], 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 one at a higher resolution, using the same initial conditions for both inputs. The generated deformation field is used in combination 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 the energy terms to be minimized to compute the deformation field $\boldsymbol{u}$:

$$\min_{\boldsymbol{u}} \; E_d(\boldsymbol{u}) + \beta_S E_{\text{smooth}}(\boldsymbol{u}) + \beta_T E_{\text{Tikhonov}}(\boldsymbol{u}), \tag{1}$$

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 $E_{\text{smooth}}$ penalizes non-smooth solutions, and the Tikhonov term penalizes vectors with large magnitudes. The discretized minimization of Eq. 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 given by:

$$\boldsymbol{A}_{\text{UpOF}} = \nabla\Phi_h{}^T\nabla\Phi_h + \beta_S\sum_j \mathbf{L}_j + \beta_T\mathbf{I}, \tag{2}$$

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 $[\nabla\Phi_h]^T\,\Phi_\delta$ where $\Phi_\delta$ is the finite difference between the input surfaces $\Phi_h$ and $\Phi_l$. 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 [2017] by its preparation and application to the input data as we use it between simulation pairs of varying resolutions (see Algo. 2 for further details).

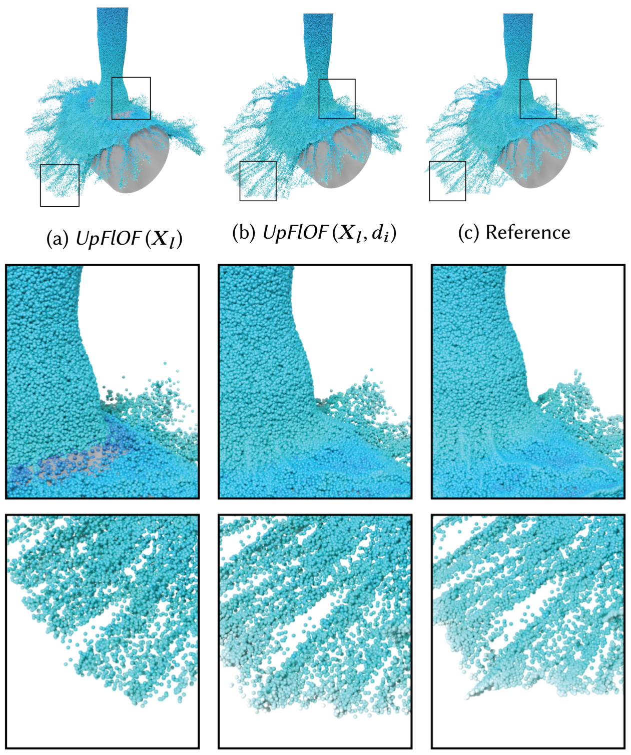


Fig. 4. Comparing the deformation precision of $\boldsymbol{u}_{\text{up}}$ when applied to a low-resolution surface using our key-event alignment term (b).

***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 $\Phi$) with topological changes. Since our method is already applying the optical solve on the SDF, we decided to use the *complex cell tests* [Wojtan et al. 2009] to determine which cells are too complex to be represented with piecewise linear isosurfaces [Varadhan et al. 2004]. 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 $\boldsymbol{x}_k$:

$$d_i = \frac{1}{d_{\boldsymbol{x}_i\to\boldsymbol{x}_k}}c_i, \tag{3}$$

where $\boldsymbol{x}_k$ is the closest key surface point 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 $\mu$ 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 $\alpha$ 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. 5). Using the proportional coefficient $d_i$, we can then rewrite Eq. 2 as follows:

$$\boldsymbol{A}_{\text{UpOF}} = \nabla\Phi_h^T\nabla\Phi_h + \beta_S\sum_j \mathbf{L}_j + \beta_T\mathbf{I} + \mathbf{D}. \tag{4}$$

As shown in Fig. 4, 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.

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).

## 4.2 Dataset and Data Augmentation

Our dataset is composed of pairs of simulated *Fluid Implicit Particle (FLIP)* liquids using initial conditions from a parameter matrix $\boldsymbol{\Theta}$. 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

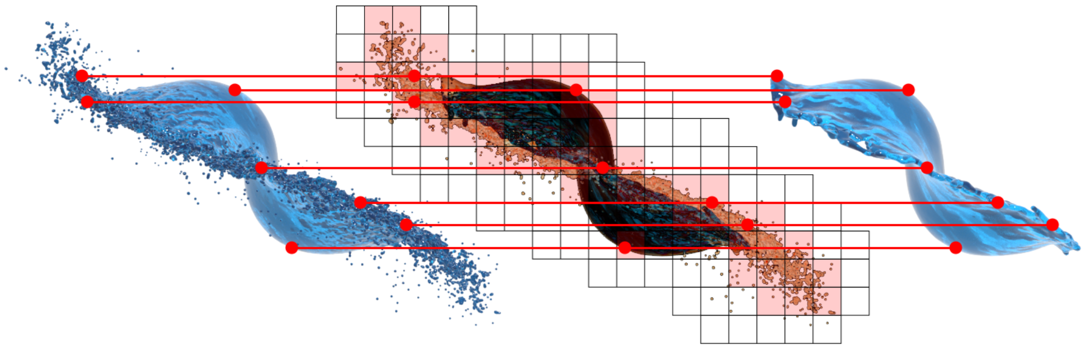


Fig. 5. As highlighted in this illustration, the key-event constraint term allows matching surface points between (right image) low- and (left image) high-resolution liquid surfaces. The complex cells are marked with a red background (middle image); the feature surface points are identified and connected using red points and lines.

**Algorithm 1:** Pseudo-code for generating and augmenting our training dataset.

```
Input: o_st: obstacle shape type (6 different shapes)      ⎫
       x_o: obstacle position ($x_o \in \mathbb{R}^3$)     ⎪
       x_em: emitter position ($x_{em} \in \mathbb{R}^3$)  ⎬ = $\Theta$
       cd: container dimensions ($cd \in \mathbb{R}^3$)   ⎭
$P = \emptyset$
for $\forall \Theta_i \in \Theta$ do                         /* for each parameter set $\Theta_i$ */
    $X_l$ = simulate($\Theta_i, ps = 0.02, gs = 2.0$)         /* FLIP solver */
    $X_h$ = simulate($\Theta_i, ps = 0.005, gs = 1.2$)        /* FLIP solver */
    $P \cup \{X_l, X_h\}$
end
$P^* = P$
for $\forall P_i \in P$ do                     /* for each pair of simulated liquids $P_i$ */
    $P_j$ = randomize($\{P_j | j \neq i, j \in 0 \leq j \leq |P|\}$)
    $\{X_l, X_h\}_i, \{X_l, X_h\}_j = P_i, P_j$
    $X_l^*$ = UpFlOF($\{X_l\}_i, \{X_l\}_j, \Phi(\{X_l\}_i), \Phi(\{X_l\}_j), \alpha_{\text{FlOF}} = 0.5$)
    $X_h^*$ = UpFlOF($\{X_h\}_i, \{X_h\}_j, \Phi(\{X_h\}_i), \Phi(\{X_h\}_j), \alpha_{\text{FlOF}} = 0.5$)
    $P^* \cup \{X_l^*, X_h^*\}$
end
```

liquids as $P_i = \{X_l, X_h\}_i$ where each sample $X_i$, 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. 1). We refer to each of these initial parameters using a subscript to our parameter matrix $\Theta \in \mathbb{R}^{n \times m}$ where $n$ is the number of parameters, and $m$ the number of initial values for each parameter (e.g., obstacle shape type $\text{o}_{\text{st}}$, obstacle position $\text{x}_{\text{o}}$, emitter position $\text{x}_{\text{em}}$, container dimensions cd). As an example, the notation $\Theta_{\text{x}_{\text{em}}}$ 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 $o_{\text{st}}$ can be either used as static boundaries or as a liquid initial shape. We present in Appendix C a small subset of generated samples for the training set.

***Data Augmentation.*** As shown in the second half of Algo. 1, we use data augmentation on the simulated liquids to artificially grow the number of samples within our dataset. In this regard, we use the original *FlOF* algorithm as proposed by Thuerey [2017] 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 $\Phi(\boldsymbol{X})$. We use $\alpha_{\text{FlOF}} = 0.5$ to generate an in-between simulation for each interpolated liquid (see the *UpFlOF* method in Algo. 2).

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.

## 5 SCENE FLOW LEARNING FOR LAGRANGIAN DEFORMATION ON PARTICLES

In this section, we will further detail the network architecture of our proposed approach (§ 5.1) and our neighborhood-based loss function for particle-based liquids (§ 5.2).

### 5.1 Network Architecture

As previously mentioned in § 3, our proposed learning architecture (Fig. 6), which we call *FFNet*, is an adaptation of the *FlowNet3D* method [Liu et al. 2019] 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 [Qi et al. 2017b]. 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.

As shown in Fig. 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 [Becker and Teschner 2007], 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:

$$\boldsymbol{f}_i^*(x) = \boldsymbol{f}_i |\boldsymbol{x} - \bar{\boldsymbol{x}}_{n_j}|, \tag{5}$$

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:

$$\bar{\boldsymbol{x}}_{n_j} = \sum_{i \in n_j} w_i \boldsymbol{x}_i. \tag{6}$$

The weights for each neighborhood $n_j$ are computed using the same smooth kernel function introduced by Zhu and Bridson [2005]:

$$w_i(x) = \frac{k(|\boldsymbol{x} - \boldsymbol{x}_i|R^{-1})}{\sum_j k(|\boldsymbol{x} - \boldsymbol{x}_j|R^{-1})}, \tag{7}$$

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.

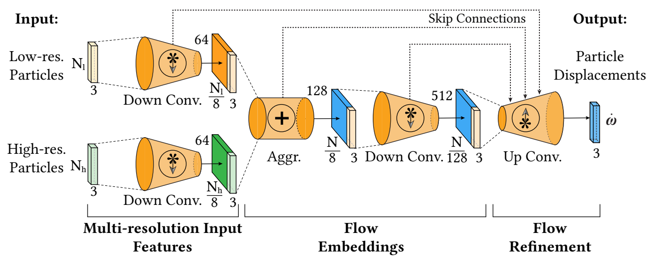


Fig. 6. The proposed network architecture. Given a pair of input liquid simulations, the particle deformation network focuses on encoding the particle displacement between low- and high-resolution particle-based simulations guided by the outputs from the *UpFlOF* approach. The dimensions at each convolution 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).

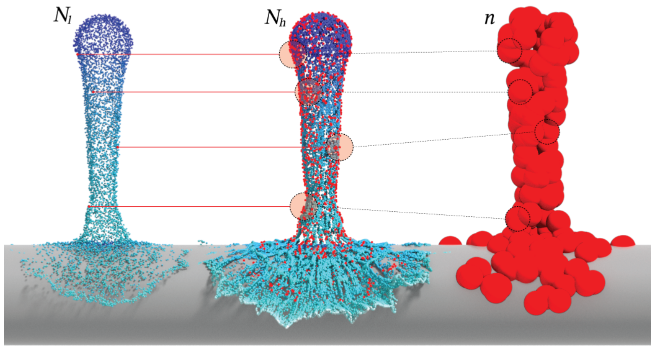


Fig. 7. At each iteration for our network, a correspondence map is generated between the input pairs of multi-resolution simulations $\boldsymbol{X}_l$ and $\boldsymbol{X}_h$ consisting of respectively $N_l$ and $N_h$ particles. These $n$ correspondences are expressed as representative neighborhood for each level of convolution.

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:

$$\boldsymbol{f}_{n_j}^* = \underset{\{i \mid \|\boldsymbol{x}_i - \boldsymbol{x}'_{n_j}\| \le R\}}{\textbf{MAX}} \{h(\boldsymbol{f}_i^*, \boldsymbol{x}_i - \boldsymbol{x}'_{n_j})\}, \tag{8}$$

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. 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 $\boldsymbol{f}_{n_j}^*$ and $\boldsymbol{g}_{n_j}^*$, 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. Again, 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 follows:

$$e_i = \underset{\{i \mid \|\boldsymbol{x}_i - \boldsymbol{x}'_j\| \le R\}}{\textbf{MAX}} \{h(\boldsymbol{f}_{n_j}^* \oplus \boldsymbol{g}_{n_j}^*, \{\boldsymbol{x}_l\}_i - \{\boldsymbol{x}_h\}_i)\}, \tag{9}$$

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

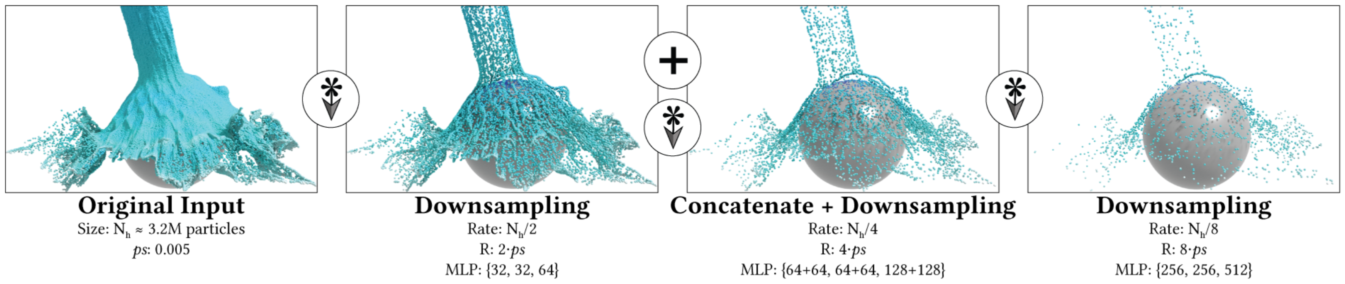


Fig. 8. Example of a cascade of downsampling convolution operations applied on a high-resolution particle-based liquid.

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. 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. 6, we use skip connections between downsampling and upsampling layers to infer multi-scale feature learning.

## 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:

$$\mathcal{L}_{\text{up}} = \frac{1}{n_j} \sum_{i}^{n_j} \|\boldsymbol{\omega}_i - \boldsymbol{\omega}_i^*\|_1 + \lambda_{n_j} \|\boldsymbol{\omega}_i^{\leftarrow} - \boldsymbol{\omega}_i\|_1, \tag{10}$$

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. 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 flow and the backflow flow (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 $\lambda_{n_j}$ 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 $\lambda_{n_j}$ using the normalized magnitude of the deformation initially computed with the *UpFlOF* algorithm (Algo. 2). 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.

## 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.

### 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 [2005]. We resample the velocity field in a MAC grid $\boldsymbol{u}_{\text{MAC}}$ that will then be updated by extrapolation within a distance $d_{\text{MAC}}$ outside the SDF $\Phi_l'$ in order to fully cover the simulation domain of the upsampled input particles $\boldsymbol{X'}$. Using the updated $\boldsymbol{u}_{\text{MAC}}$, 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 $\boldsymbol{u}'_{\text{MAC}}$ 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. 2 (with *isInference()* equal to *true*).

**Algorithm 2:** *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
$\Phi_l$ = computeSDF($\boldsymbol{X}$)
$\boldsymbol{X}'$ = resampleNarrowBand($\boldsymbol{X}, d_b$)
if isInference() then                              /* for predicting displacements */
    $\dot{\boldsymbol{\omega}}$ = predict($\boldsymbol{X}$)                              /* see Fig. 6 */
    $\boldsymbol{u}_\omega$ = transferToGrid($\dot{\boldsymbol{\omega}}$)
    $\Phi'_l = \Phi_l$
else                                               /* for data augmentation */
    $\Phi'_l$ = upscaleInterpolate($\Phi_l, r_h$)
    $\boldsymbol{u}_\omega$ = FlOF($\Phi'_l, \Phi_h, \alpha_{\mathrm{FlOF}} = 1$)      /* see Algo. 1 in [Thuerey 2017] */
end
$\hat{\boldsymbol{u}}_\omega$ = resampleOFtoMAC($\boldsymbol{u}_\omega, \boldsymbol{u}_{\mathrm{MAC}}$)
$\boldsymbol{u}_{\mathrm{MAC}}$ = extrapolate($\boldsymbol{u}_{\mathrm{MAC}}, \Phi'_l, d_{\mathrm{MAC}}$)
$\boldsymbol{u}'_{\mathrm{MAC}} = \|\hat{\boldsymbol{u}}_\omega\| \boldsymbol{u}_{\mathrm{MAC}}$
$\hat{\boldsymbol{u}}'_\omega = \hat{\boldsymbol{u}}_\omega + \boldsymbol{u}'_{\mathrm{MAC}}$                 /* inject input motion */
$\boldsymbol{X}_{\mathrm{advect}}$ = advectInGrid($\boldsymbol{X}', \hat{\boldsymbol{u}}'_\omega, t$)         /* $t = t + \triangle t$ */
```

## 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. [2019], 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* [Ferstl et al. 2016] 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. 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 noticeable (especially in motion). Also, as exposed in Fig. 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.

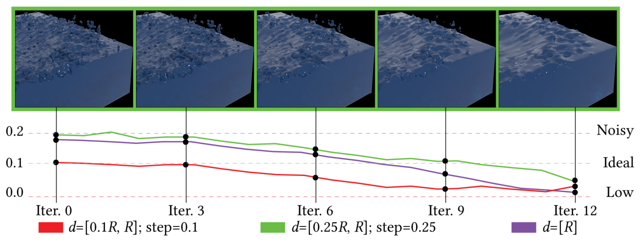


Fig. 9. We compare the influence of varying upsampling distances $d$ over the surface noise (expressed as an error on the vertical axis) generated throughout the inference iterations. The images corresponding to the evolving level of noise (i.e., from iteration 1 to 12) of the green curve is presented on the top of the chart.

## 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.

### 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. 1. We have divided the datasets into three categories referred to as *Colliding*, *Shape*, and *Container* (see Table 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. 18 in Appendix C). 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 $o_{\mathrm{st}}$, 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 [Thuerey 2017] on shape's SDF, shape position, and emission position. On both *Colliding* and *Shape* datasets, we interpolated with weight $\alpha_{\mathrm{FlOF}} = 0.5$, 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 $\alpha_{\mathrm{FlOF}} \in \{0.25, 0.50, 0.75\}$ giving us an augmentation factor of $\times 8$. Finally, we used a data split of

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

Table 1. Datasets of simulation pairs used as training and validation sets. We also present the augmentation factor (Aug. factor) used to grow our datasets prior to training.

90%:10% respectively for training and validation sets. The test set was composed of new and significantly different simulation setups as enumerated in Table 2. We also provide a breakdown of the evaluation times of the presented examples in Appendix B.

***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 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.** | **Sim.** | ***FFNet* eval.** |
|---|---|---|---|---|
| *Pouring*: Fig. 1 | 500k | $128^3$ | 0.724 | 0.071 |
| *Collision*: Fig. 5 | 320k | $96^3$ | 0.542 | 0.061 |
| *Cylinder*: Fig. 4 | 290k | $96^3$ | 0.497 | 0.055 |
| *Stirring*: Fig. 12 | 250k | $128^3$ | 0.482 | 0.058 |
| *Multi-stage*: Fig. 13 | 1100k | $192^3$ | 1.249 | 0.112 |
| *Multi-streams*: Fig. 17 | 900k | $192^3$ | 1.045 | 0.098 |

Table 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 [Kingma and Ba 2015] 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. 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 in Appendix A 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** | **Conv. time** |
|---|---|---|---|
| *PointNet* [Qi et al. 2017a] | 0.45 | 26.11% | 14.4 |
| *PointNet++* [Qi et al. 2017b] | 0.44 | 29.84% | 13.6 |
| *FlowNet3D* [Liu et al. 2019] | 0.37 | 52.27% | 10.3 |
| *FFNet* (ours) | **0.23** | **63.71%** | **8.1** |

Table 3. Flow estimation results on the *Colliding* dataset.

As highlighted in Table 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. 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 § 7.2), the EPE reduces significantly.

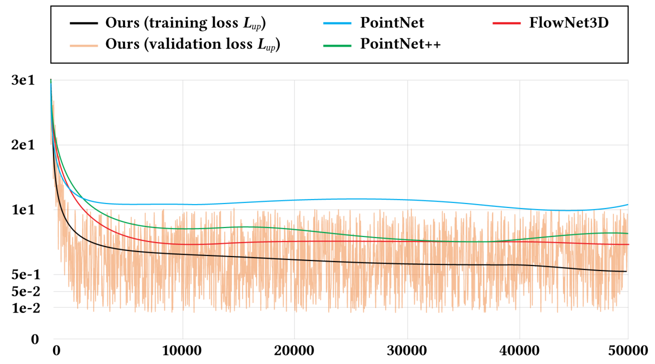


Fig. 10. Comparing training losses on the *Container* dataset measured with our approach and a few point-based learning baselines. In order to lighten the chart, the validation loss (noisy orange curve) is showed only for our approach (black curve).

Fig. 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 mimic the high-resolution details with good accuracy. Fig. 10 also highlights the generalization capabilities on a much more complex example (Fig. 17). 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. 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.

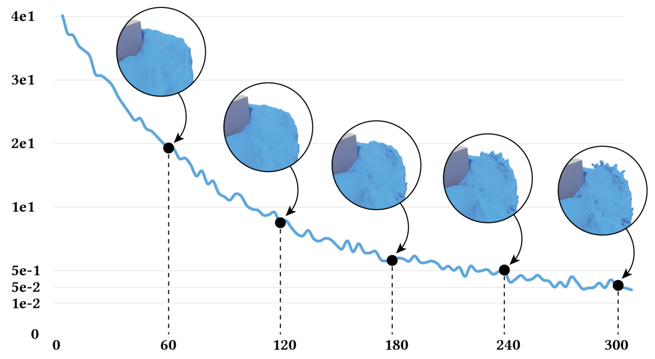


Fig. 11. Convergence plot of our $\mathcal{L}_{\mathrm{up}}$ loss for the *Stirring* example from Fig. 12. We show the loss error ($Y$-axis) evolving throughout the training epochs ($X$-axis).

## 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 $\boldsymbol{u}_{\mathrm{up}}$ (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 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.

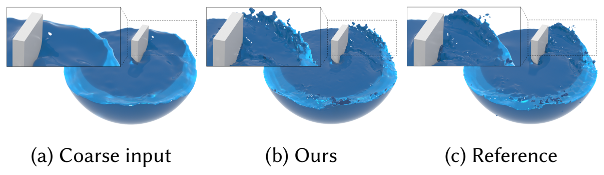


Fig. 12. Comparison of our result (b) generated with *FFNet* using a low-resolution input (a) with the corresponding high-resolution reference (c).

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. [2018]. For example, as exposed in the close-up insets of

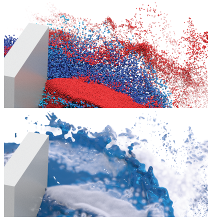


Fig. 12, our approach generates convincing splashing details while mostly remaining attached to the main volume of liquid. On the other hand, our network successfully reproduced eddies surrounding the moving obstacle that stirs the liquid. However, as compared with the high-resolution reference (Fig. 12c), we were unable to recreate the small droplets detached from the splashing portions of the liquid. 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

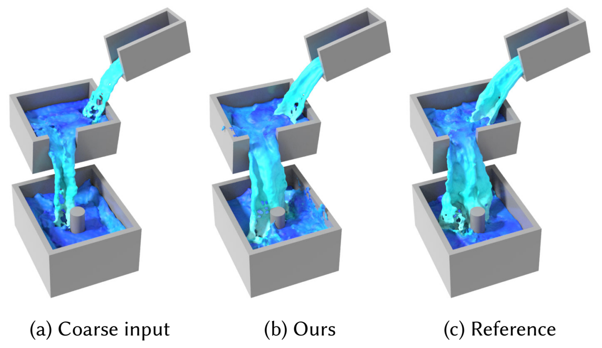


Fig. 13. Comparison of our result (b) generated with *FFNet* using a low-resolution input (a) with the corresponding high-resolution reference (c).

so on). We provide an example to highlight this limitation in the supplemental video.

Fig. 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 § 7.1, our dataset is composed of solely single sources of disturbance either on static liquid or static obstacles. In Fig. 17, 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. 17, the energy loss caused by a coarse discretization (Fig. 17a) is partially restored in appearance when up-resed using our approach (Fig. 17c). The arced streamlines in our result (Fig. 17c) closely reproduce those of the high-resolution reference (Fig. 17b). It is also apparent in Fig. 13 that the static container is significantly more agitated in our result than it is in the coarse 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. 13b) was showing a slightly different but plausible turbulence behavior (Fig. 13c).

***Improved Alignment.*** The alignment term (Eq. 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. 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. 14 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).

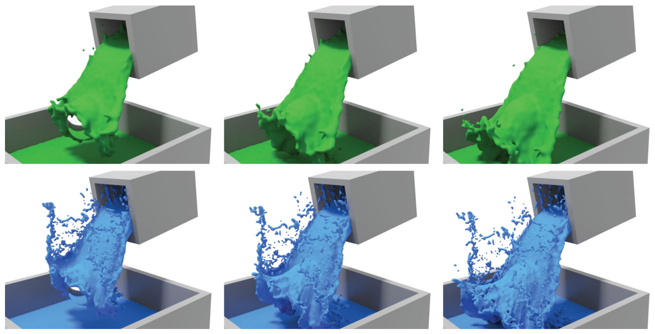


Fig. 14. Comparing the evolution between our inferred liquid (top) and the high-resolution reference (bottom).

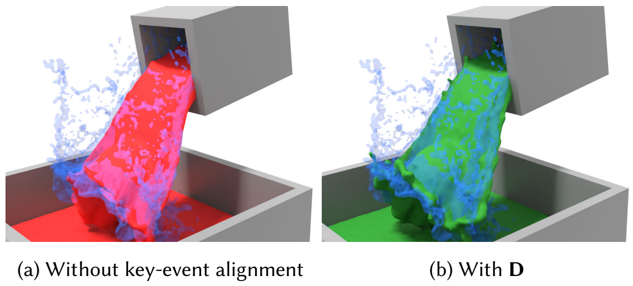


Fig. 15. Comparing before and after applying the proposed key-event alignment term on the teaser example (Fig. 1).

The benefit of using key-event alignment is illustrated in Fig. 15, 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 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

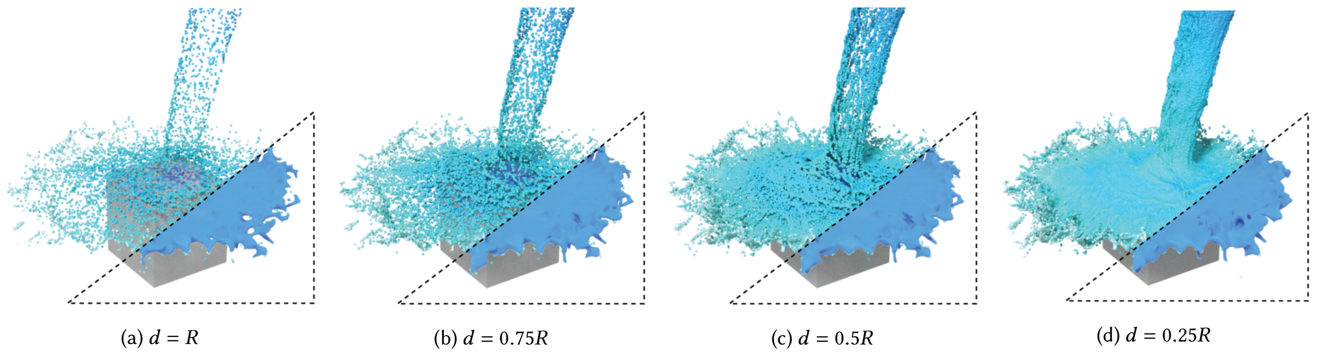


Fig. 16. We show the influence of upsampling the particle band prior to the inference step. As shown in this figure, we qualitatively evaluated the results by varying the resampling radius of the narrow band.

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. 13 and 17 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. 16, 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 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. [2019], 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.

## 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. [2018], 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, the Université de Montréal, and the École de Technologie Supérieure (ÉTS). We would also like to give a special thanks to Nils Thuerey for his insightful advice, comments, and discussions surrounding this project during the research internship at the Technical University of Munich (TUM) at the very beginning of this project. Finally, We would like to thank Autodesk Inc. for providing resources to carry out this project in the end.

## REFERENCES

Mridul Aanjaneya, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. 2017. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Trans. on Graphics (TOG)* 36, 4, Article 140 (2017), 12 pages.

Kfir Aberman, Peizh Uo Li, Dani Lischinski, Olga Sorkine-Hornung, Daniel Cohen-Or, and Baoquan Chen. 2020. Skeleton-aware networks for deep motion retargeting. *ACM Trans. on Graphics (TOG)* 39, 4, Article 62 (2020), 14 pages.

Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J Guibas. 2007. Adaptively sampled particle fluids. *ACM Trans. on Graphics (TOG)* 26, 3, Article 48 (2007), 8 pages.

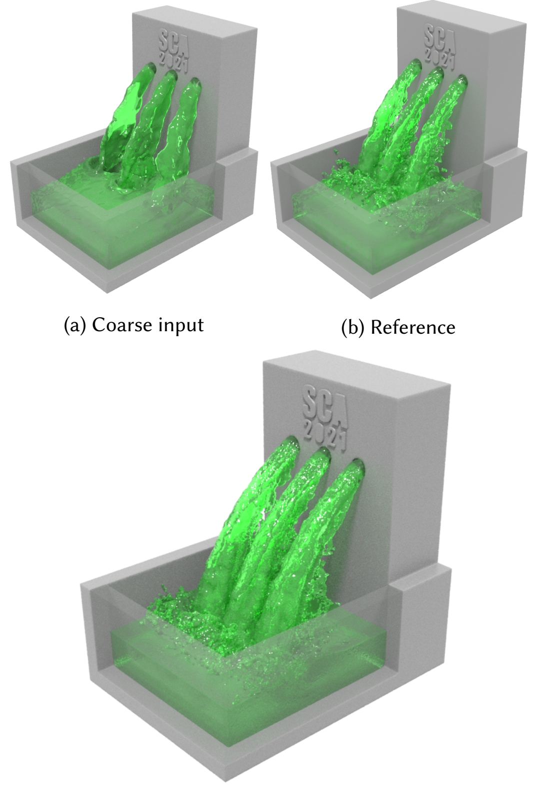
Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. 2012. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Trans. on Visualization and Computer Graphics* 18, 8 (2012), 1202–1214.

Ryoichi Ando, Nils Thürey, and Chris Wojtan. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. on Graphics (TOG)* 32, 4, Article 103 (2013), 10 pages.

Markus Becker and Matthias Teschner. 2007. Weakly compressible SPH for free surface flows. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*. 209–217.

Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. on Graphics (TOG)* 36, 4, Article 98 (2017), 12 pages.

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



(a) Coarse input (b) Reference

(c) Ours

Fig. 17. Example highlighting the generalization capacities of our network in an unknown simulation setup.

Mengyu Chu and Nils Thuerey. 2017. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Trans. on Graphics (TOG)* 36, 4, Article 69 (2017), 14 pages.

Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A Efros, and Adrien Bousseau. 2018. 3D sketching using multi-view deep volumetric prediction. *Proc. ACM on Computer Graphics and Interactive Techniques* 1, 1, Article 21 (2018), 22 pages.

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

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

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

Theodore Kim, Jerry Tessendorf, and Nils Thuerey. 2013. Closest point turbulence for liquid surfaces. *ACM Trans. on Graphics (TOG)* 32, 2, Article 15 (2013), 13 pages.

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

L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-driven fluid simulations using regression forests. *ACM Trans. on Graphics (TOG)* 34, 6, Article 199 (2015), 9 pages.

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

Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim, and Derek Nowrouzezahrai. 2015. Surface turbulence for particle-based liquid simulations. *ACM Trans. on Graphics (TOG)* 34, 6, Article 202 (2015), 10 pages.

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

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

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

Karthik Raveendran, Chris Wojtan, and Greg Turk. 2011. Hybrid smoothed particle hydrodynamics. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*. 33–42.

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

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

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

Barbara Solenthaler and Markus Gross. 2011. Two-scale particle simulation. *ACM Trans. on Graphics (TOG)* 30, 4, Article 81 (2011), 8 pages.

Nils Thuerey. 2017. Interpolations of smoke and liquid simulations. *ACM Trans. on Graphics (TOG)* 36, 1, Article 3 (2017), 16 pages.

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

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

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

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

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

Tuanfeng Y Wang, Hao Su, Qixing Huang, Jingwei Huang, Leonidas J Guibas, and Niloy J Mitra. 2016. Unsupervised texture transfer from images to model collections. *ACM Trans. on Graphics (TOG)* 35, 6, Article 177 (2016), 13 pages.

Zirui Wang, Shuda Li, Henry Howard-Jenkins, Victor Prisacariu, and Min Chen. 2020. FlowNet3D++: Geometric losses for deep scene flow estimation. In *Proc. IEEE/CVF Winter Conference on Applications of Computer Vision*. 91–98.

Maximilian Werhahn, You Xie, Mengyu Chu, and Nils Thuerey. 2019. A multi-pass GAN for fluid flow super-resolution. *Proc. ACM on Computer Graphics and Interactive Techniques* 2, 2, Article 10 (2019), 21 pages.

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

Rene Winchenbach, Hendrik Hochstetter, and Andreas Kolb. 2017. Infinite continuous adaptivity for incompressible SPH. *ACM Trans. on Graphics (TOG)* 36, 4, Article 102 (2017), 10 pages.

Rene Winchenbach and Andreas Kolb. 2021. Optimized refinement for spatially adaptive SPH. *ACM Trans. on Graphics (TOG)* 40, 1, Article 8 (2021), 15 pages.

Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. 2009. Deforming meshes that split and merge. *ACM Trans. on Graphics (TOG)* 28, 3, Article 76 (2009), 10 pages.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Trans. on Graphics (TOG)* 37, 4, Article 95 (2018), 15 pages.

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

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

## 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. Specifications of the MLP layers used in our network architecture.

## 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 2, we present a breakdown of the computation times (in seconds) under *FFNet eval.* as shown in Table 5.

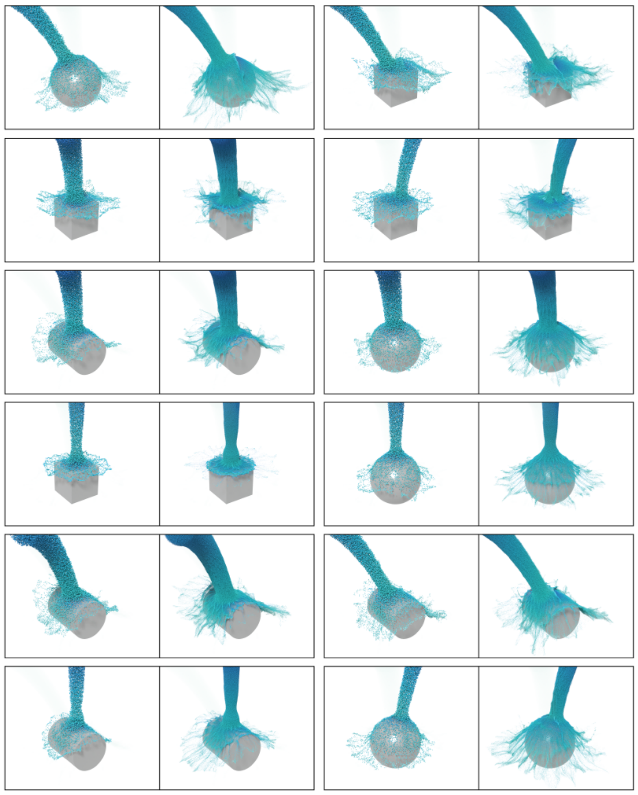
| **Example** | **Ref.** | ***FFNet* eval.** | | | |
|---|---|---|---|---|---|
| | | Part. Upsampl. | Network Predict. | *UpFlOF* Advect. | **Speed-up** |
| Fig. 1 | 6.9 | 0.012 | 0.043 | 0.016 | 16× |
| Fig. 5 | 5.2 | 0.010 | 0.038 | 0.013 | 15× |
| Fig. 4 | 4.9 | 0.009 | 0.034 | 0.012 | 13× |
| Fig. 12 | 4.8 | 0.009 | 0.036 | 0.013 | 17× |
| Fig. 13 | 11.1 | 0.020 | 0.067 | 0.025 | 19× |
| Fig. 17 | 10.3 | 0.017 | 0.059 | 0.023 | 17× |

Table 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.

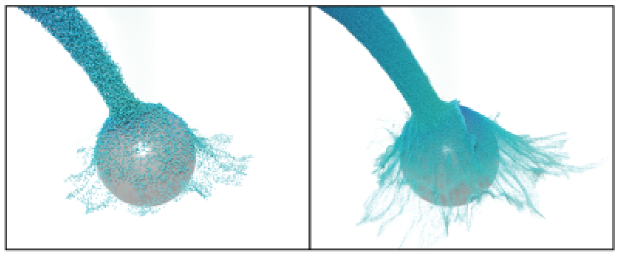
The *FFNet eval.* step includes the particle upsampling, the displacement predictions by our network, and the advection using the *UpFlOF* advection scheme. Each step has been implemented using CUDA capabilities to offer interactive computation times. Both upsampling and advection steps are really fast 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.

## 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.



(a) Small subset



(b) Multi-resolution simulation pair

Fig. 18. (a) A small subset of simulation pairs from our multi-resolution *Colliding* dataset used to train the proposed *FFNet* model. (b) One expanded pair to better display similarities and differences.

In Fig. 18, we present a small subset of our datasets (a). A sample pair is presented in a close-up view (b) to better highlight the subtle differences between in the input resolutions.

## D NOTATION

We provide here a table to clarify the notation used in this paper. Generally speaking, bold uppercases are used for matrices, bold lowercases for vectors, and italics for single values. A dotted field is actually for one particle of that field.

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

Table 6. Notation and associated meaning for our terms.