Liquid Splash Modeling with Neural Networks

KIWON UM, Technical University of Munich
XIANGYU HU, Technical University of Munich
NILS THUEREY, Technical University of Munich

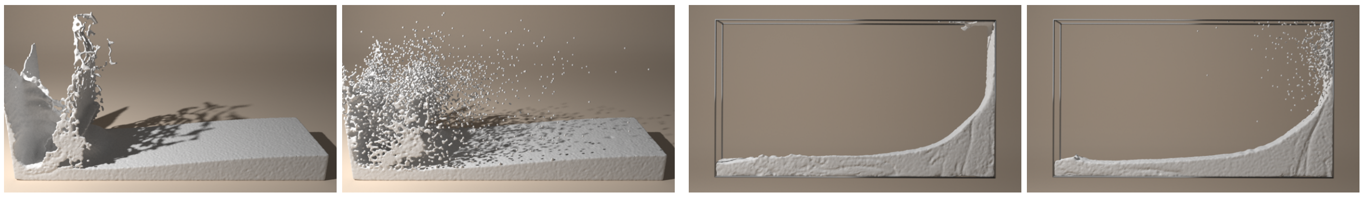
(a) breaking dam (b) sloshing wave

Fig. 1. Our data-driven splash model improves the visual fidelity of the basic FLIP simulation in two examples: a breaking dam and sloshing wave simulations. Each example shows the visual comparisons between FLIP (left) and a simulation employing our model (right).

This paper proposes a new data-driven approach for modeling detailed splashes for liquid simulations with neural networks. Our model learns to generate small-scale splash detail for fluid-implicit-particle methods using training data acquired from physically accurate, high-resolution simulations. We use neural networks to model the regression of splash formation using a classifier together with a velocity modification term. More specifically, we employ a heteroscedastic model for the velocity updates. Our simulation results demonstrate that our model significantly improves visual fidelity with a large amount of realistic droplet formation and yields splash detail much more efficiently than finer discretizations. We show this for two different spatial scales and simulation setups.

Additional Key Words and Phrases: machine learning, neural network, liquid simulation, fluid implicit particle, smoothed particle hydrodynamics

1 INTRODUCTION

For large-scale liquid simulations, it is crucial for visual fidelity that a numerical simulation can produce sufficient amounts of very small droplets. However, it is difficult to capture such splashes in practical simulations due to their small-scale structure. Simulations of detailed structures typically require the use of very fine spatial discretizations, which in turn lead to high computational cost. Thus, it is often challenging to generate vivid splashes in liquid simulations because these require resolving the small-scale dispersive motions that lead to droplets forming and being ejected from the bulk volume.

This paper proposes a new data-driven splash model that improves the visual fidelity of hybrid particle-grid liquid simulations. By learning the formation of small-scale splashes from physically accurate simulations, our model effectively approximates the sub-grid scale effects that lead to droplets being generated. This enables us to generate realistic splashes in coarse simulations without the need for manually tweaking parameters or increased computational costs induced by high-resolution simulations. We realize our model using machine learning techniques with neural networks (NNs) and integrate the model into the fluid-implicit-particle (FLIP) algorithm. Figure 1 shows the outcome of our model, which we will denote with MLFLIP in the following.

This work is supported by the *ERC Starting Grant* 637014.

2 RELATED WORK

The behavior of liquids is typically modeled as *Navier-Stokes* equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \mathbf{g} - \frac{1}{\rho}\nabla P + \nu \nabla^2 \mathbf{u} \quad \text{and} \quad \nabla \cdot \mathbf{u} = 0, \tag{1}$$

where $\mathbf{u}$ is the velocity, $\mathbf{g}$ is the gravity, ρ is the density, P is the pressure, and ν is the viscosity coefficient. There exist many numerical methods for solving these equations. Those solvers are commonly categorized as Eulerian and Lagrangian approaches (Bridson 2015; Ihmsen et al. 2014).

FLIP is a particularly popular method for liquid simulations (Zhu and Bridson 2005), and it is the most widely used one in movie visual effects at the moment. Its effective combination of Lagrangian and Eulerian properties enables the efficient solve of liquid motions. While FLIP has become a practical solution for liquid simulations, the core method has been extended in various ways to improve its simulation quality and efficiency. For example, different position correction methods were introduced to improve the distribution of particles (Ando et al. 2012; Um et al. 2014). In addition, Ferstl et al. (2016) proposed a narrow band method that improves the efficiency by sampling the volume with particles only near the surface.

The goal of our model is to improve the visual fidelity of liquid simulations with splash detail. Apart from FLIP, the smoothed particle hydrodynamics (SPH) approach is a popular alternative in graphics (Müller et al. 2003; Solenthaler and Pajarola 2009). Ihmsen et al. (2012) proposed a flexible model for secondary particle effects for SPH simulations. With enough manual tuning, such a secondary particle method can yield realistic results, but in contrast to their work, we focus on an automated approach that captures splash effects for physically-parametrized real world scales. Our model does not require any parameter tuning on the user side. At the same time, one of our goals is to demonstrate that neural networks are a suitable tool to detect and generate these splashes.

A method that shares our goal to enable splashes with FLIP is the unilateral incompressibility solver (Gerszewski and Bargteil 2013). While it also aims at letting FLIP particles disperse more easily, our approach targets a very different direction. Instead of modifying the pressure solve, we incorporate a statistical model with the help of machine learning. Given a trained model, our approach is easily integrated into existing solvers.

Machine learning: As we employ a machine learning technique for our splash model, we will also briefly review previous work on machine learning. In general, the learning process aims for the approximation of a general function f using a given data set (i.e., input $\mathbf{x}$ and output $\mathbf{y}$) in the form of $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$ where $\mathbf{w}$ is the set of weights and biases to be trained. Using NNs, the general function f is modeled by networks of multiple layers where each layer contains multiple nodes (i.e., artificial neurons). These networks consist of layers with connected nodes. The output vector $\mathbf{y}_L$ from a layer L is typically computed with $\mathbf{y}_L = \Phi(\mathbf{w}_L\mathbf{y}_{L-1} + \mathbf{b}_L)$ where $\Phi(\cdot)$ is the activation function that is applied to each component, $\mathbf{w}_L$ is the weight matrix of the layer, and $\mathbf{b}_L$ is the bias vector of the layer. Here, the activation function Φ makes it possible to capture non-linearities in the approximated function. We will demonstrate that the NNs technique, which so far have rarely been used for fluid simulations, can be used for realization of our data-driven splash model.

Such NNs were previously used to compute local pressure approximations (Yang et al. 2016) while others employed networks with convolutional layers to regress the whole pressure field (Tompson et al. 2016). Moreover, an approach using regression forests, which represent an alternative to neural networks, was proposed to efficiently compute forces in SPH (Ladický et al. 2015). More recently, NNs were also successfully employed for patch-based smoke simulations (Chu and Thuerey 2017) and fast generation of liquid animations with space-time deformations (Bonev et al. 2017). In the engineering community, approximating effects smaller than the discretization resolution is known as *coarse graining* (Hijón et al. 2009), but this idea has not been used to model splash formation. We propose to use machine learning techniques to represent accurate and high-resolution data in order to approximate complex small-scale effects with high efficiency. To the best of our knowledge, splash modeling using machine learning techniques has not been studied before.

3 DATA-DRIVEN SPLASH MODELING

The following section details our data-driven approach for generating splashes. The principal idea of our approach is to infer statistics about splash formation based on data from simulations that are parametrized to capture the droplet formation in nature. Our definition of a *splash* is a small disconnected region of liquid material that is not globally coupled with the main liquid volume. Thus, we treat individual splashing droplets as particles that only experience gravitational acceleration but no other NS forces. This modeling is in line with the secondary particle effects often employed in movies (Losure and Baer 2012). The key novelty of our approach is that it does not require manually chosen parameters, such as velocity or curvature thresholds, to generate the splashes. Rather, it uses a statistical model and data extracted from a series of highly detailed and pre-computed simulations.

Our approach consists of two components: a *detachment classification* and a *velocity modification* step. Based on a feature descriptor consisting of localized flow information, the classifier predicts whether a certain volume of liquid forms a detached droplet within a chosen duration Δt (typically, on the order of a single frame of animation). For droplets that are classified as such, our modifier predicts its future velocity based on a probability distribution for velocity modifications. We use NNs to represent both components, and the following sections describe our statistical model and the corresponding machine learning approach.

3.1 Neural Network Model

The input to our model is a feature descriptor $\mathbf{x} \in \mathbb{R}^M$ that encapsulates enough information of the flow such that a machine learning model can infer whether the region in question will turn into a splash. For this binary decision "*splash* or *no-splash*", we will use an indicator value $l \in \{1, 0\}$ in the following. Each descriptor is typically generated for a position $\mathbf{p}$. The M individual components of $\mathbf{x}$ consist of flow motion and geometry in the vicinity of $\mathbf{p}$. In practice, we use 3^3 samples of the velocity and level set. The discussion of this choice will be given in Section 3.3 in more detail.

We train our models with a given data set that consists of feature vectors $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ and corresponding detachment indicator values $\mathbf{L} = \{l_1, l_2, \cdots, l_N\}$; they are generated during a pre-processing phase at locations $\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$. Then, our classifier aims for inferring the probability P_s that a feature vector $\mathbf{x}_i$ is in the class indicated by l_i. Considering a probability distribution function $\mathbf{y}_s$ that P_s follows, we will approximate the function $\mathbf{y}_s$ from the given data. For this task, we can follow established procedures from the machine learning field (Bishop 2006).

The probability distribution $\mathbf{y}_s(\mathbf{x}_i, \mathbf{w}_s)$ is the target function that is represented by the weights $\mathbf{w}_s$. The weights are the actual degrees of freedom that will be adjusted in accordance to the data during the learning phase. We can express P_s in terms of $\mathbf{y}_s$ as:

$$P_s(l_i|\mathbf{x}_i) \sim P\left(l_i|\mathbf{y}_s(\mathbf{x}_i, \mathbf{w}_s)\right), \tag{2}$$

which yields the following likelihood function that we wish to maximize:

$$L_d(\mathbf{L}|\mathbf{X}) = \prod_{i=1}^{N} P\left(l_i|\mathbf{y}_s(\mathbf{x}_i, \mathbf{w}_s)\right). \tag{3}$$

In order to maximize this likelihood, we use the well-established *softmax* (i.e., normalized exponential function) for the loss of our classification networks. A successfully trained model will encode $\mathbf{y}_s$; thus, we can evaluate with new feature vectors at any position in a flow to predict whether the region will turn into a detached droplet within the time frame Δt.

Let $\Delta\mathbf{v}$ be an instance of a velocity change for a splash with respect to the motion of the bulk liquid in its vicinity. We will afterward consider this velocity change of a droplet relative to the bulk as the *velocity modification* of a particle in our simulation. Similar to the classifier above, our goal is to infer the set of velocity modifications $\Delta\mathbf{V} = \{\Delta\mathbf{v}_1, \Delta\mathbf{v}_2, \cdots, \Delta\mathbf{v}_N\}$ based on the corresponding set of feature vectors $\mathbf{X}$. From the statistics of our training data, we found

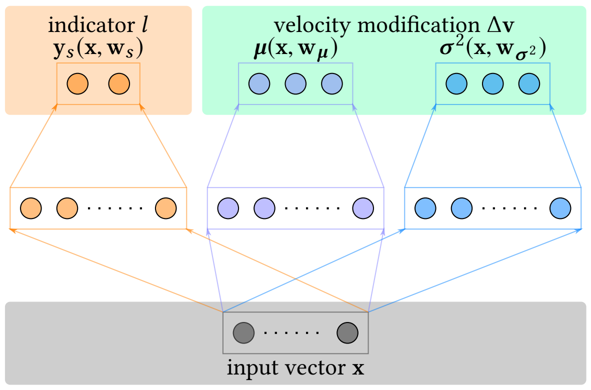
indicator l
$\mathbf{y}_s(\mathbf{x}, \mathbf{w}_s)$
velocity modification $\Delta\mathbf{v}$
$\boldsymbol{\mu}(\mathbf{x}, \mathbf{w}_\mu)$
$\boldsymbol{\sigma}^2(\mathbf{x}, \mathbf{w}_{\sigma^2})$
input vector $\mathbf{x}$

Fig. 2. The overall structure of our neural networks.

that it is reasonable to assume that the velocity modifications follow a normal distribution relative to the mean flow direction. Accordingly, we model the modifier as a modification function $f_m(\Delta\mathbf{v}_i|\mathbf{x}_i)$ following a normal distribution with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$:

$$f_m(\Delta\mathbf{v}_i|\mathbf{x}_i) \sim \mathcal{N}\left(\Delta\mathbf{v}_i|\boldsymbol{\mu}(\mathbf{x}_i, \mathbf{w}_\mu), \boldsymbol{\sigma}^2(\mathbf{x}_i, \mathbf{w}_{\sigma^2})\right), \qquad (4)$$

thus,

$$f_m(\Delta\mathbf{v}_i|\mathbf{x}_i) \sim \frac{1}{\sqrt{2\pi\boldsymbol{\sigma}_i^2}} \exp\left(\frac{-(\Delta\mathbf{v}_i - \boldsymbol{\mu}_i)^2}{2\boldsymbol{\sigma}_i^2}\right), \qquad (5)$$

where, for the sake of simplicity, $\boldsymbol{\mu}_i$ and $\boldsymbol{\sigma}_i^2$ denote $\boldsymbol{\mu}(\mathbf{x}_i, \mathbf{w}_\mu)$ and $\boldsymbol{\sigma}^2(\mathbf{x}_i, \mathbf{w}_{\sigma^2})$, respectively. Then, the negative log likelihood function L_m (also known as loss function) for the given data is defined as follows:

$$L_m(\Delta\mathbf{V}|\mathbf{X}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{d}\left[\frac{(\Delta\mathbf{v}_{i,j} - \boldsymbol{\mu}_{i,j})^2}{\boldsymbol{\sigma}_{i,j}^2} + \ln\boldsymbol{\sigma}_{i,j}^2\right], \qquad (6)$$

where j denotes the spatial index.

As Equation (6) indicates, we model the modifier as a mean variance estimation (MVE) problem (Khosravi et al. 2011; Nix and Weigend 1994). Instead of directly estimating the mean of targets, the MVE problem assumes that errors are normally distributed around the mean and estimates both the mean and *heteroscedastic* variance. Note that $\boldsymbol{\mu}(\mathbf{x}_i, \mathbf{w}_\mu)$ and $\boldsymbol{\sigma}^2(\mathbf{x}_i, \mathbf{w}_{\sigma^2})$ are the target functions that are approximated with each set of weights $\mathbf{w}_\mu$ and $\mathbf{w}_{\sigma^2}$.

In our approach, the mean and variance functions are represented by NNs and approximated by estimating the two sets of weights such that they minimize the loss function L_m for the given data $\{\mathbf{X}, \Delta\mathbf{V}\}$. We want to point out that several machine learning algorithms are available to solve this problem (Bishop 2006), but NNs have proven themselves to be robust in this problem, and we found that they work sufficiently well in our tests.

The overall structure of our NNs is illustrated in Figure 2. The NNs learn for two separate components: classifier and modifier. Sharing the input vector $\mathbf{x}$, both components are represented as separate two-layer NNs. The size of output from the first layer is double the input vector's, and the output is fully connected to the final output. All outputs from each layer are activated using the hyperbolic tangent function. Note that there is a large variety of different network layouts that could tried here, but we found that this simple structure worked sufficiently well in our tests.

We realized our NNs approach with the *TensorFlow* framework (2016) using its ADAM optimizer (Kingma and Ba 2014) with a learning rate of 10^{-4}. We also employ weight decay and dropout (both with strength 10^{-1}) to avoid over-fitting. Additionally, we found that the batch normalization technique (Ioffe and Szegedy 2015) significantly improves the learning rate and accuracy.

3.2 FLIP Integration

Our NN-based splash generation model easily integrates into an existing FLIP simulation pipeline; we will denote the new FLIP method integrated with our splash model as *MLFLIP*. After the pressure projection step, we run classification on all particles in a narrow band of one cell around the surface. For those that generate a positive outcome, we evaluate our velocity modification network to compute component-wise mean and variance. Then, we generate random numbers from correspondingly parameterized normal distributions to update the velocity of the new splash particle. All splashes are treated as ballistic particles and do not participate in the FLIP simulation until they impact the main volume again.

Look-ahead correction: While our splash generation algorithm reliably works in our tests, we found that a small percentage of particle, about 0.05-0.5%, can be misclassified. This can happen, for instance, when the side of an obstacle is just outside the region of our input feature vector. To minimize the influence of such misclassifications, we implemented a look-ahead step that reverts the classification of individual splashes if necessary. For this look-ahead check, we advance all bulk volume particles, i.e., those that were not classified as splashes, by Δt using the current grid velocities. We separately integrate positions of the new splash particles for a time interval of Δt with their updated velocities. If a new splash particle ends up inside of the bulk liquid or an obstacle, we revert its classification and modification.

3.3 Training Data

Our NNs require a large set of input feature vectors with target outputs for training. In our model, the latter consists of the classification result l indicating whether a flow region forms a splash and the velocity modification $\Delta\mathbf{v}$ predicting the trajectory of a splash. We generate the training data from a series of simulations with randomized initial conditions designed to incite droplet formation. The randomized initial conditions are the number of droplets and their initial positions and velocities. We choose the ranges of each condition such that they yield sufficient variability for data generation. The snapshots of the randomized example simulations are shown in Figure 3. Note that at this stage any available "accurate" NS solver could be employed. However, we demonstrate that FLIP can *bootstrap* itself by using high-resolution simulations with correctly parameterized surface tension.

For the training data simulations, it is crucial that they resolve important sub-grid effects that are probably not resolved on coarse resolutions to which our model will be applied later. We test our model with two small-scale physics where the surface tension is dominant thus generates many droplets. Our training simulations

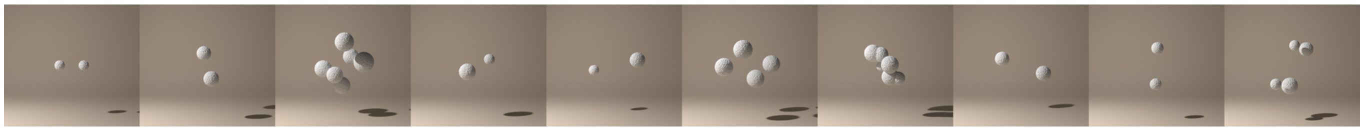
Fig. 3. Ten randomized initial conditions to generate the droplet formation data for training.

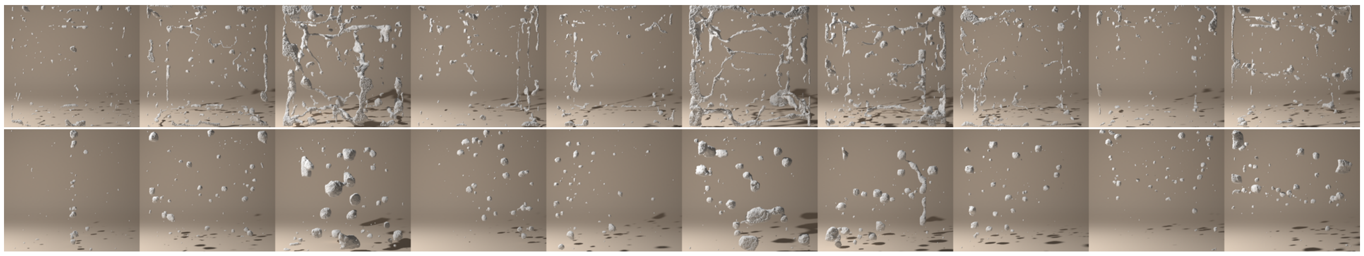
Fig. 4. Example frames of the 1cm (top) and 0.3cm (bottom) scale simulations for training.

are performed using FLIP with the ghost fluid method (Hong and Kim 2005) for surface tension effects. The two scales use a grid spacing of 1cm and 0.3cm, and they are parametrized with the surface tension of water, i.e., 0.073 N/m. Both scales use a simulation grid of 100^3. Several example frames for both scales are shown in Figure 4. Our networks successfully extract the dynamics of droplet formation based on coarse data and thus effectively model small-scale dynamics that would be costly to resolve with a regular simulation.

Note that the simulation data for training will be used to encode the desired sub-grid effects for much larger scales afterwards. When applying our model in new simulation setups, we typically have scales that are much coarser than those used for generating the training data. Thus, the feature descriptor (i.e., $\mathbf{x}_i$) needs to be defined at this coarse simulation scale, and our networks need to infer their outputs (i.e., l_i and $\Delta\mathbf{v}_i$) based only on this coarse input. For this purpose, we make use of a coarse grid at data generation time. This coarse grid represents the scale to which the model will be applied afterward. For every time step, we down-sample the necessary high-resolution fields from the data generation simulation to this lower resolution and extract the feature descriptors for training.

As indicated in Section 3.1, we define a feature descriptor $\mathbf{x}_i$ using 3^3 samples of the velocity and level set values interpolated with a sampling spacing of h; i.e., the feature vector consists of 108 components containing $3^3 \times 3$ velocity values and $3^3 \times 1$ level set values. Because the splash formation mostly relies on the local flow physics

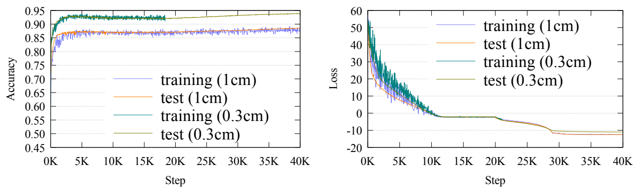

Fig. 5. Learning graphs for both the training and test data sets in two scales. The left graph shows the classification accuracy; the right graph shows the loss L_m.

near the liquid surface, we focus on the localized flow information and the surface region where the splash is likely initiated. From pilot experiments, we found that the improvement is negligible when more samples or more features such as obstacle information are used for the feature vector.

In order to extract the splash indicator value l, we analyze the particle's spatial motion for a chosen duration Δt (i.e., a single frame of animation in our experiments). Using an auxiliary grid, the separate volumes are recognized by computing the isolated liquid regions from the level set field or cell flags. We then identify the splashing particles (i.e., l=1) as those ending up in a new disconnected component that falls below a given volume threshold at time $t + \Delta t$. In our case, if a disconnected component consists of less than 8 cells, the volume is marked as droplet volume. All particles in such a droplet are marked as splashing if the droplet did not exist as a disconnected component at time t.

The velocity modifier of our model predicts the trajectory for a splash. We evaluate this prediction after updating the velocity of particle from the divergence-free velocity. Thus, the new velocity $\mathbf{v}_m^{t+\Delta t}$ for a splash particle is defined as $\mathbf{v}_m^{t+\Delta t} = \mathbf{v}^{t+\Delta t} + \Delta\mathbf{v}$, and we compute the velocity modification $\Delta\mathbf{v}$ for training with:

$$\Delta\mathbf{v} = \frac{\mathbf{p}^{t+\Delta t} - \mathbf{p}^t}{\Delta t} - \mathbf{v}^{t+\Delta t}. \tag{7}$$

The splashes are initiated near the liquid surface in general; thus, we extract the training data only from the surface particles. The surface particles are recognized by slightly expanding the area of empty (or air) cells. Note that we use the data from splashing as well as non-splashing particles as training data. For the latter, we set $\Delta\mathbf{v} = 0$. It is crucial for training that the networks see sufficiently large amounts of samples from both classes.

For training, we used 500K samples of the 1cm scale and 102K samples of the 0.3cm scale. They were extracted from ten training simulations per scale. The data contain the same number of both splashing and non-splashing samples. We randomly split them into 75% for the training data set and 25% for the test data set. The graphs in Figure 5 illustrate the progress of the learning phase in

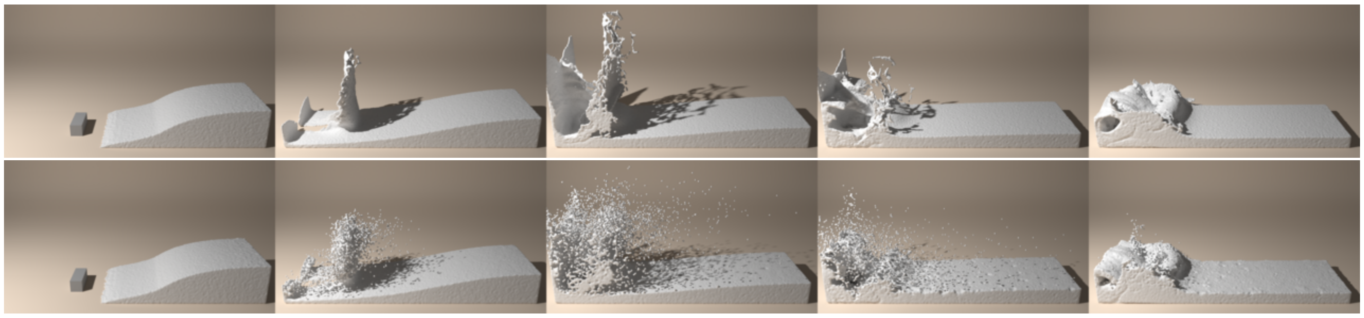
Fig. 6. Example frames of breaking dam simulations with FLIP (top) and MLFLIP (bottom).

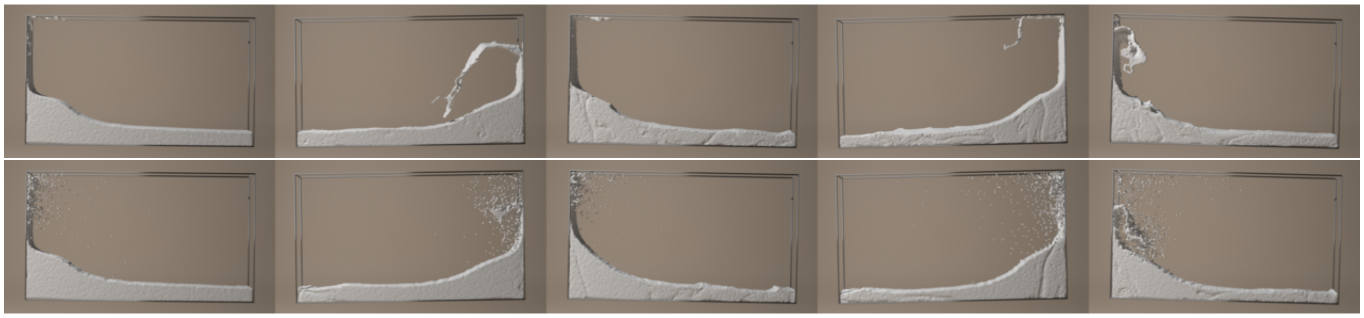
Fig. 7. Example frames of sloshing wave simulations with FLIP (top) and MLFLIP (bottom).

our experiments. The training performed for 40K iterations with 5K samples as a training batch. When the data were fully used, we randomly shuffled the samples and then continued with the training.

4 RESULTS

This section demonstrates that our data-driven splash model (i.e., MLFLIP) improves the visual fidelity of liquid simulations for two examples: a breaking dam and sloshing wave tank setups. The former should represent ca. three meters of real world size, while the sloshing wave tank is ca. one meter in length. Hence, we use the 1cm splash model for the breaking dam and 0.3cm splash model for the sloshing wave. Figure 6 shows the comparisons between FLIP and MLFLIP for the breaking dam example. Our model leads to a significant increase in violent and detailed splashes for this large scale flow. Despite the large number of splashes, the plausibility of the overall flow is preserved. Likewise, our model robustly introduces splashes also in the smaller sloshing wave example as shown in Figure 7. The smaller real world size in conjunction with the smaller velocities leads to fewer splashes being generated for this setup.

Our model requires additional calculations for generation of the splashes, and consequently, this results in an additional increased runtime. In the breaking dam example, the average computation time per simulation step is 0.5s for FLIP, while it is 1.6s for MLFLIP. Both simulations use the same grid resolution of 160×150×50. In the sloshing wave example, it is 0.2s for FLIP and 0.5s for MLFLIP. In this case, the grid resolution is 150×84×10. The breaking dam setup requires ca. 5 simulation steps per frame of animation, while the sloshing wave requires ca. 8 on average.

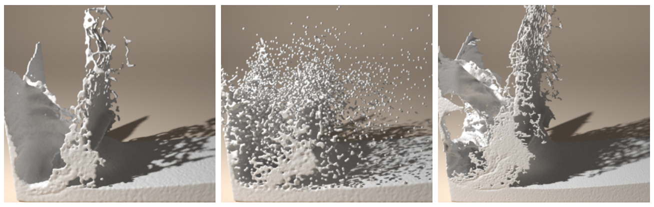
Fig. 8. Comparisons among the three simulations: FLIP (left), MLFLIP (middle), and FLIP with the high resolution (right).

The most time consuming part of MLFLIP is the evaluation step for the classifier and modifier. In the breaking dam example, its computation time is 1.0s on average out of the 1.6s per simulation step. The evaluation step leaves significant room for optimizations as we simply employ the CPU evaluation of the NN library here. As GPUs are very well suited for evaluating NNs, we expect that our MLFLIP would not impose a significant overhead compared to a regular FLIP simulation when evaluating the model on GPUs. The cost for computing feature vectors is negligible with 0.03s per simulation step because it only affects a narrow band near the surface.

We observed that the splashes of our MLFLIP simulation are very difficult to achieve with regular FLIP simulations even with high resolutions. For the breaking dam example, Figure 8 shows a visual comparison of three simulations: FLIP and MLFLIP with the same resolution and FLIP with a doubled resolution of 320×300×100. Despite taking 6.5 times longer per frame of animation (i.e., 54s for

FLIP and 8.3s for MLFLIP), this high resolution simulation fails to resolve the splashing effects of our MLFLIP simulation. Thus, our model successfully generates splash details from a low resolution simulation, while the high resolution simulation barely improves the amount of detail despite its high computational cost.

5 CONCLUSIONS AND FUTURE WORK

This paper introduces a new data-driven splash model that is realized using machine learning techniques with NNs. We demonstrate that our NNs and training process successfully learn the formation of splashes at a certain physical scale from our training data. Our model leads to improved splashing liquids and thus successfully extracts the relevant mechanisms for droplet formation from the pre-computed data.

So far, we only experimented with two different spatial scales. The training data were generated independently for both scales, and the neural networks were trained separately. However, we envision that our model could be enlarged and trained with data from a variety of spatial scales leading to a more generic model that could be applicable to a broad range of targeted real world scales. Additionally, other complex effects such as bubbles, capillary waves, and foam could be incorporated into our model in the future.

REFERENCES

2016. TensorFlow. (2016). http://tensorflow.org/ *http://tensorflow.org/.*

Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. 2012. Preserving Fluid Sheets with Adaptively Sampled Anisotropic Particles. *IEEE Transactions on Visualization and Computer Graphics* 18, 8 (2012), 1202–1214. DOI: https://doi.org/10.1109/TVCG.2012.87

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Boris Bonev, Lukas Prantl, and Nils Thuerey. 2017. Pre-computed Liquid Spaces with Generative Neural Networks. *arXiv preprint* (2017).

Robert Bridson. 2015. *Fluid Simulation for Computer Graphics.* CRC Press.

Mengyu Chu and Nils Thuerey. 2017. Data-driven Synthesis of Smoke Flows with CNN-based Feature Descriptors. *ACM Trans. Graph.* 36, 4 (2017).

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.

Dan Gerszewski and Adam W. Bargteil. 2013. Physics-Based Animation of Large-Scale Splashing Liquids. *ACM Trans. Graph.* 32, 6 (Nov. 2013), 185:1–185:6. DOI: https://doi.org/10.1145/2508363.2508430

Carmen Hijón, Pep Español, Eric Vanden-Eijnden, and Rafael Delgado-Buscalioni. 2009. Mori-Zwanzig Formalism as a Practical Computational Tool. *Faraday Discussions* 144 (Oct. 2009), 301–322. DOI: https://doi.org/10.1039/B902479B

Jeong-Mo Hong and Chang-Hun Kim. 2005. Discontinuous Fluids. *ACM Trans. Graph.* 24, 3 (July 2005), 915–920. DOI: https://doi.org/10.1145/1073204.1073283

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

Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. 2014. SPH Fluids in Computer Graphics. In *Eurographics 2014 - State of the Art Reports.* Eurographics Association, Strasbourg, France, 21–42. DOI: https://doi.org/10.2312/egst.20141034

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]* (Feb. 2015). arXiv:cs/1502.03167

A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya. 2011. Comprehensive Review of Neural Network-Based Prediction Intervals and New Advances. *IEEE Transactions on Neural Networks* 22, 9 (Sept. 2011), 1341–1356. DOI: https://doi.org/10.1109/TNN.2011.2162110

Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv:cs/1412.6980

Lubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-Driven Fluid Simulations Using Regression Forests. *ACM Trans. Graph.* 34, 6 (Oct. 2015), 199:1–199:9. DOI: https://doi.org/10.1145/2816795.2818129

Jeff Budsberg Michael Losure and Ken Museth Matt Baer. 2012. Liquids in The Croods. *SIGGRAPH Talks* (2012).

Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-Based Fluid Simulation for Interactive Applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '03).* Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 154–159.

D. A. Nix and A. S. Weigend. 1994. Estimating the Mean and Variance of the Target Probability Distribution. In *IEEE International Conference on Neural Networks, 1994. IEEE World Congress on Computational Intelligence*, Vol. 1. 55–60. DOI: https://doi.org/10.1109/ICNN.1994.374138

B. Solenthaler and R. Pajarola. 2009. Predictive-corrective Incompressible SPH. *ACM Trans. Graph.* 28, 3, Article 40 (July 2009), 6 pages. DOI: https://doi.org/10.1145/1531326.1531346

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2016. Accelerating Eulerian Fluid Simulation With Convolutional Networks. *arXiv preprint: 1607.03597* (2016).

Kiwon Um, Seungho Baek, and JungHyun Han. 2014. Advanced Hybrid Particle-Grid Method with Sub-Grid Particle Correction. *Computer Graphics Forum* 33, 7 (Oct. 2014), 209–218. DOI: https://doi.org/10.1111/cgf.12489

Cheng Yang, Xubo Yang, and Xiangyun Xiao. 2016. Data-Driven Projection Method in Fluid Simulation. *Computer Animation and Virtual Worlds* (Jan. 2016), 415–424. DOI: https://doi.org/10.1002/cav.1695

Yongning Zhu and Robert Bridson. 2005. Animating Sand As a Fluid. *ACM Trans. Graph.* 24, 3 (July 2005), 965–972. DOI: https://doi.org/10.1145/1073204.1073298