Expert Systems With Applications 152 (2020) 113410

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

FluidsNet: End-to-end learning for Lagrangian fluid simulation

Yalan Zhang[a], Xiaojuan Ban[a,*], Feilong Du[a], Wu Di[b]

[a] Beijing Advanced Innovation Center for Materials Genome Engineering, University of Science and Technology, Beijing, China
[b] Norwegian University of Science and Technology, Norway

ARTICLE INFO

Article history:
Received 16 August 2019
Revised 24 January 2020
Accepted 24 March 2020
Available online 2 April 2020

Keywords:
deep learning
Lagrangian fluid simulation
physical-based animation

ABSTRACT

Over the past few decades, fluid simulation has emerged as an important tool in computer animation. However, traditional physical-based fluid simulation systems are time consuming and requires large computational resources to generate large-scale fluid flows. Intelligent systems provide us a new method of data-driven to accelerate simulations. Previous intelligent system manually crafted feature vectors by using a context-based integral method and resulted in high computational and memory requirements. Unlike the existing techniques, we do not use any manually crafted feature, instead directly operate on Lagrangian fluid simulation data and use machine learned features. This paper presents a novel end-to-end deep learning neural network that can automatically generate a model for fluid animation based-on Lagrangian fluid simulation data. This approach synthesizes velocity fields with irregular Lagrangian data structure using neural network. Every fluid particle is treated independently and identically. We use symmetric functions to capture space structure and interactions among particles and design different network structures to learn various hierarchical features of fluid. We test this method using several data sets and applications in various scenes with different sizes. Our experiments show that the model is able to infer velocity field with realistic details such as splashes. In addition, compared with exiting simulation system, this method shows significant speed-ups, especially on large scene simulations.

1. Introduction

Physically-based modeling has been an important research topic in computer graphics. The requirement for modeling and simulating of real world is reflected in many fields such as movie special effects, video games, industrial production, virtual reality, etc. In the film manufacturing industry, some shots, such as floods and tsunami, are too expensive to produce in the real world through real scene shooting. At this time, these scenes have to be simulated and realized by using computer special effects technology. In addition, computer animation technology can also create scenes that do not exist in the real world, allowing people to realize and express their infinite imagination. 3D modeling, light rendering techniques and physics-based simulation techniques are used to bring the audience an immersive feeling.

With the development of simulation technology and breakthroughs in computer computing capabilities, people have increasingly demands on the scale and visual effects of simulation scenes. However, Traditional fluid simulation systems is time consuming and requires large computational resources. Besides, for large-scale simulation scenes such as lakes and oceans, it is impossible to obtain highly realistic simulation in real-time. On the other hand, thanks to the development of machine learning technology, some graphics researches use intelligent systems to utilize data-driven methods to accelerate simulation. In this field, deep learning has achieved remarkable progress on Eulerian fluid simulation in recent years, including speed up (Tompson, Schlachter, Sprechmann, & Perlin, 2016; Yang, Yang, & Xiao, 2016), capture the intricate details of turbulent flows (Chu & Thuerey, 2017; Xie, Franz, Chu, & Thuerey, 2018), and predict temporal evolution (Wiewel, Becher, & Thuerey, 2018), *etc.* However, few works exist (Ladický, Jeong, Solenthaler, Pollefeys, & Gross, 2015) for Lagrangian fluid simulation. The reason behind is deep learning methods require highly regular input data formats but Lagrange simulation data is a set of 3D points which are fundamentally irregular. In fact, Euler and Lagrange methods concentrate on different aspects of fluid. This study aims to apply deep learning in Lagrange fluid. There are two main notable motivations. First it can depict more microscopic details in simulation process. Second, it is more suitable for complex simulation scenarios.

Deep learning for physical simulation is a critical topic in intelligent system areas. Its applications have already covered from a long-term predictor for the trajectory of the object (Ehrhardt, Monszpart, Mitra, & Vedaldi, 2017), to a robust control for imitating the

* Corresponding author.
E-mail addresses: yalan.zhang920503@gmail.com (Y. Zhang), banxj@ustb.edu.cn (X. Ban), dufldylan@gmail.com (F. Du), di.wu@ntnu.no (W. Di).

https://doi.org/10.1016/j.eswa.2020.113410
0957-4174/

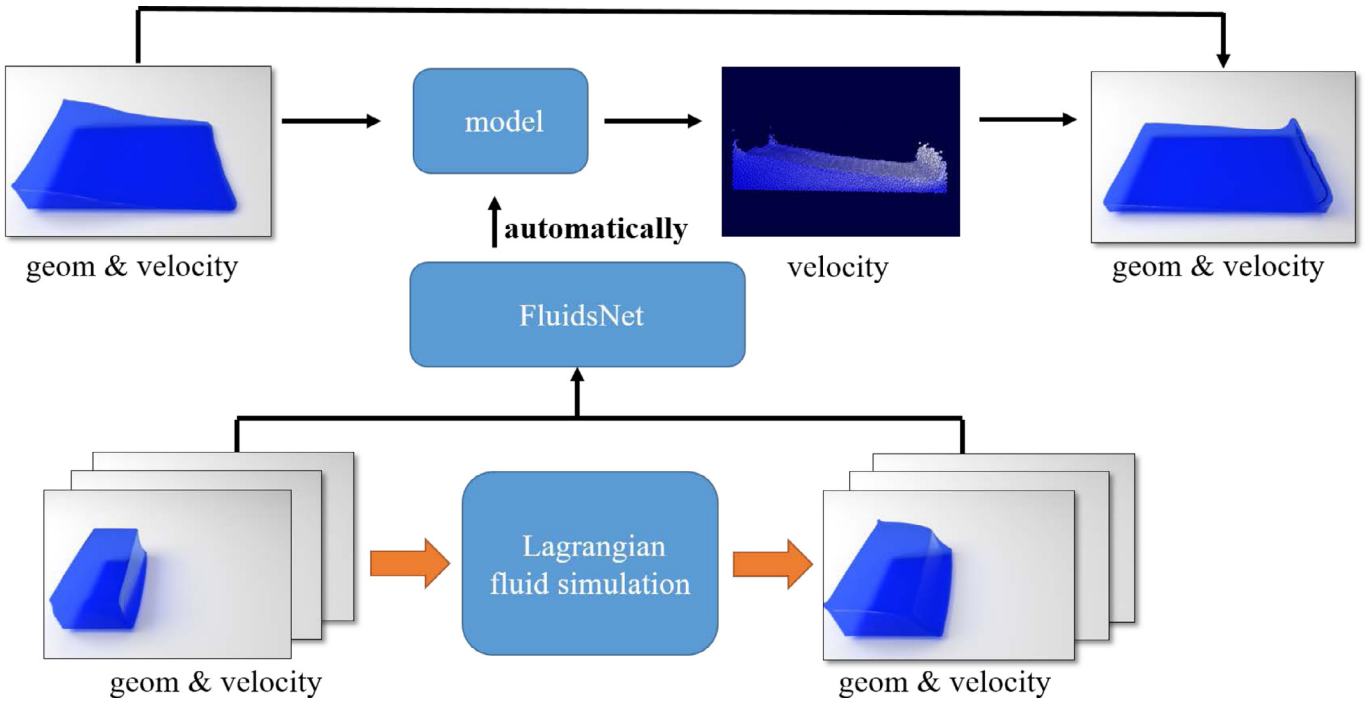


Fig. 1. Our end-to-end deep learning neural network learns to automatically find a model to generate fluid animation based-on Lagrangian fluid simulation data.

reference motions (Peng, Abbeel, Levine, & van de Panne, 2018). Although the physical simulations are effective for approximation of dynamics systems, it requires high cost to solve physical models with Partial Differential Equations (PDEs). intelligent systems can help to improve the efficiency for the equation solving process. At the same time, the massive high quality data brought by the simulation system can help the intelligent system to directly generate more precise models.

In this paper, we introduce a new end-to-end intelligent system, FluidsNet, for Lagrangian fluid simulation. It uses Lagrangian fluid simulation data as input to automatically find a model to generate fluid animation, as shown in Fig. 1. We believe that it is a particular interesting challenge to combine Lagrangian simulation system with deep learning, as it can not only lead to stable and efficient fluid simulation, but also could be useful for giving intelligent systems predictive capabilities for other simulation systems based on Lagrange description. E.g., giving intelligent systems the ability to learn and understand the Lagrangian system, could help us to forecast traffic conditions (Kim, Son, Tian, Chiu, & Yang, 2017; Wilkie, Sewall, & Lin, 2013), or even simulate large-scale emergencies and develop strategies (Higo, Okada, Hipel, & Fang, 2017; Wagner & Agrawal, 2014).

The Lagrangian methods, (*e.g.*: Smoothed Particle Hydrodynamics (Lucy, 1977) (SPH)), approximate continuous quantities in the incompressible Navier-Stokes equations by using discrete moving particles. Since fluid particles are continuously distributed in the space without specific order, it is necessary to make a model invariant to arrange the inputs. Besides, as fluid particles are not isolated, the model must be able to capture space structure and interactions among particles, and then learn the dynamics of fluid. In the initial stages of FluidsNet, each fluid particle is processed identically and independently. The local feature is generated to describe the relationship between one particle and its neighbors. Then FluidsNet divides the simulation scenario into equally spaced 3D voxels and encodes them. After that, 3D convolution aggregates local voxel representations and transforms them into a global feature. The final fully connected layers of the network aggregate the independent feature, local feature and global feature to predict the velocity field of every particle.

The key contributions of our work are as follows:

- A novel intelligent system suitable for consuming Lagrangian fluid simulation data to generate fluid animations automatically;
- An efficient feature learning network that directly operates on fluid particles to avoid information bottlenecks introduced by manual design features;
- A thorough empirical and detailed analysis on the stability and efficiency of the proposed approach.

2. Related work

2.1. Deep learning & fluids

Using neural network for fluid simulation is an attractive research direction. Exiting works mainly target on learning Eulerian fluid simulation. For a single timestep, convolutional networks are used to speed up the projection step (Tompson et al., 2016; Yang et al., 2016). Chu and Thuerey (2017) used a convolutional network on smoke synthesis by replacing a low-precision area with pre-existing small scale details. Xie et al. (2018) used a generative adversarial network to generate highly detailed smokes based on a low-resolution field. Bonev, Prantl, & Thuerey (2017) used generative neural networks to synthesize the deformation of a liquid surface. Resent works have also addressed the problem of temporal and spatial evolution of simulation. Wiewel et al. (2018) predicted the changes of the pressure field over time with long short-term memory (LSTM) units. Kim et al. (2018) synthesized fluid velocities continuously in space and time by using a few parameters for generation. Other work used deep reinforcement learning to control 2D coupled fluid/rigid animations (Ma, Tian, Pan, Ren, & Manocha, 2018). However, all of these methods are only applicable to Euler grid data and cannot be directly used to Lagrangian simulation data.

Closer to our line of work, Ladický et al. (2015) proposed a regression forests method to compute local interactions of particle-based liquids. A major limitation of their method is that they manually craft a feature vector for each fluid particle by using a context-based integral method. As the feature vector needs to be rich enough to encode all necessary information required to predict the next state, for every particle it would compute about 50,000 features at each time step. Since typical simulation scenarios usually have at least 50,000 fluid particles, training the architecture would result in high computational and memory requirements. In contrast to their work, our intelligent system directly operates on Lagrangian fluid simulation data and avoids information bottlenecks introduced by manual feature engineering. Both our intelligent system and their intelligent system can provide reasonable predictions and robust simulations for Lagrangian fluid simulation. However, their model requires computer clusters for training and running, while ordinary GPUs is enough for our intelligent system.

2.2. Deep learning & physics.

Recently, deep learning algorithms begin to influence physical modeling by learning and predicting directly from experiments where the underlying causal model is complicated or unknown. In the field of spatio-temporal statistics, Cressie and Wikle (2015) showed how to build statistical models by using physical background knowledge. Battaglia, Hamrick, and Tenenbaum (2013) proposed a cognitive mechanism model based on intuitive physics to simulate physical phenomena by using approximate probabilities. Mottaghi, Bagherinezhad, Rastegari, and Farhadi (2016) designed a Newtonian Neural Network that learns to map a single image to a state in a Newtonian scenario to predict the forces and the motion of objects in the image. Lerer, Gross, and Fergus (2016) and Li, Leonardis, and Fritz (2016) both presented a learning-based approach based on the pictures of simulated data that predicted stability of towers comprised of wooden blocks under different conditions and the trajectory of the wooden block after the toy tower collapses. Some works such as Agrawal, Nair, Abbeel, Malik, and Levine (2016) or Denil et al. (2016) attempted to model the dynamics of a robot's interactions based on deep neural networks to learn the intuitive physics of objects. Other works (Battaglia, Pascanu, Lai, Rezende et al., 2016; Chang, Ullman, Torralba, & Tenenbaum, 2016; Ehrhardt et al., 2017; Watters et al., 2017) have modeled the motion of Lagrangian objects based on real data to explore the ability of deep learning in order to mine potential physical models. However, in this category of research, there are only a handful of objects in the scene that are studied for physical motion, such as the trajectory of a small ball (Ehrhardt et al., 2017), or the collision between two or three objects (Chang et al., 2016; Watters et al., 2017). In our work, the fluid particles in the scene are usually above 50K, and each is independently calculated but correlated with others. Therefore, although we also need physics learning, learning fluid dynamics, the above research models cannot be directly used to solve our problems.

2.3. Deep learning & irregular inputs

As our method focuses on the learning of irregular Lagrangian data sets, we also give a brief overview of the related work here, with a particular focus on 3D point clouds.Qi, Su, Mo, and Guibas (2017) proposed PointNet to achieve input order invariance by using symmetric function over inputs. The follow-up works (Qi, Yi, Su, & Guibas, 2017; Wang et al., 2018)] further improved the quality of extracted point features by considering the local structures in point clouds. Li, Bu, Sun, and Chen (2018) proposed a convolution method for point clouds to aggregate information from a neighborhood into less and less points, but each with richer information. Other methods tried to transfer irregular data to a regular format and applied existing deep learning algorithms to the adapted structure.Zhou and Tuzel (2017) proposed VoxelNet to learn features by subdividing the 3D space into equally spaced voxels and performing convolution operations on it. Recent space partition methods like k-d trees (Klokov & Lempitsky, 2017) or octrees (Wang, Liu, Guo, Sun, & Tong, 2017) remedied some resolution issues but still relied on the subdivision of a bounding volume. Different from our work, these methods only focused on static 3D shape detection, and did not involve interactions and dynamic changes within the points/particles. Despite some inspirations on feature learning for irregular data, our method aims to learn fluid dynamics from fluid data and generate fluid animation automatically.

3. Problem statement

3.1. Deep learning for lagrangian-based fluid simulation

In the traditional Lagrangian fluid simulation method, continuous quantities are approximated by using discrete moving particles, as shown in Algorithm 1. In each physics update, it needs to find the local neighbors k for all particles $P : \mathbb{R}^{N\times 3} \mapsto \mathbb{R}^{N\times 3\times k}$, where N is the number of particles in the space. Then, the density, pressure, and the resulting forces acting on each particle are evaluated and used to compute accelerations, which we use function$F : (\mathbb{R}^{N\times \mathrm{m}}, P) \mapsto \mathbb{R}^{N\times 3}$ to present, where m is the feature numbers of every particle. Finally, particle physical quantities of the next state are integrated using the estimated accelerations, which could be presented as $I : (\mathbb{R}^{N\times \mathrm{m}}, F) \mapsto \mathbb{R}^{N\times m}$.

Algorithm 1: The basic Lagrangian-based fluid simulation.

while *animating* **do**
  forall the *particle i* **do**
    find neighbor particles k_i
  end
  forall the *particle i* **do**
    $\rho_i = \sum_j m_j W_{ij}$
    compute p_i use ρ_i
  end
  forall the *particle i* **do**
    $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i}\nabla p_i$
    $\mathbf{F}_i^{viscosity} = m_i \mathbf{v}\nabla^2 \mathbf{v}_i$
    $\mathbf{F}_i^{surface} = \frac{m_i}{\rho_i}\sigma\nabla^2\mathbf{x}$
    $\mathbf{F}_i^{other} = m_i\mathbf{g}$
    $\mathbf{a}_i(t) = (\mathbf{F}_i^{pressure} + \mathbf{F}_i^{viscosity} + \mathbf{F}_i^{surface} + \mathbf{F}_i^{other})/m_i$
  end
  forall the *particle i* **do**
    $\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \Delta t\mathbf{a}_i(t)$
    $\mathbf{x}_i(t+\Delta t) = \mathbf{x}_i(t) + \Delta t\mathbf{v}_i(t+\Delta t)$
  end
end

Our approach aims for automatically finding the best model to generate fluid animation.

Given a set of fluid particles with physical quantities on the spatial domain $\Omega \subset \mathbb{R}^3$, we want to find the velocity field from the data. According to the Navier-Stokes equations for incompressible fluids:

$$\frac{d\mathbf{v}}{dt} + \mathbf{v}\cdot\nabla\mathbf{v} = \frac{1}{\rho}(-\nabla p + \mu\nabla^2\mathbf{v} + \sigma\nabla^2\mathbf{x} + \mathbf{g}),$$

we assume that the observed data are associated with a generic nonlinear evolution function

$$\mathbf{v}_i = \mathrm{F}^{'}(\mathbf{x}, \Delta\mathbf{x}, \mathbf{v}, \Delta\mathbf{v}, \ldots), \mathbf{x} \in \Omega \subset \mathbb{R}^3, i \in [1, N].$$

To infer $F^{'}$, we design a deep neural network (DNN), named FluidsNet, taking $\mathbb{R}^{N\times m}$ as input and $\mathbb{R}^{N\times 3}$ as output. In this paper, we take position, velocity and a fluid/solid marker as input feature, the input Lagrangian fluid simulation data is denoted by $\mathbf{P} = \{\mathbf{p}_i = [x_i, y_i, z_i, u_i, v_i, w_i, s/f] \in \mathbb{R}^7, i \in [1, N]\}$.We take velocity field as output feature, $F^{'} \in \mathbb{R}^{N\times 7} \rightarrow \mathbb{R}^{N\times 3}$. The FluidsNet structure is parameterized by its weights W and biases b, determines a function space $\mathcal{F}$ and aims to find the optimal solution $F^{'} \in \mathcal{F}$. For now, we will assume that all functions are continuous and sufficiently differentiable.

We have two goals when calculating the velocity field: the main goal is staying true to the original simulations by $\min_{F^{'}\in\mathcal{F}} (F^{'} - F)^2 + \|W\|_2$, where we use the mean squared error to measure the distance between the two functions and$\|W\|_2$ is a L^2 norm biases term. As for the other goal, we would like to make the FluidsNet has considerable performance on speed-up large-scale scene simulation. Specifically, as long as the size ratio of the large scene to the small scene is the same (for example, the aspect ratio of 1:1:1), the trained model from a small scene could be used to predicate the output of a large scene by modifying the voxel size proportionally.

3.2. Challenges

Learning features is one of our main challenges in this work. While Ladický et al. (2015) manually crafted feature vectors by using a context-based integral method, we use machine learned features instead of hand-crafted features to model fluid information. The restriction to irregular input data, such as a set of fluid particles, yields several requirements for the machine-learned features. First, since Lagrangian fluid simulation data is a set of particles without specific order, our FluidsNet need to be invariant to input permutation. Second, as fluid particles are not isolated, the feature needs to model the behavior of other particles in both a close or far neighborhood. Furthermore, the evaluation of the feature should be possible in constant time with normal resources demand, resulting in a running time linear with the number of particles. To make the feature rich enough to model all possible constellations of neighboring particles, hand-crafted features would contain lots of redundant information and result in a high requirement for computing and memory resources. Instead of manually trying to find heuristics or approximations of how these physical features might be designed and influence predictions, we transfer this task to a machine learning algorithm by our FluidsNet. Inspired by Qi, Su, et al. (2017), the machine-learned features can handle the unordered input issue by using symmetry functions such as multi-layer perceptron (MLP) and pooling layers. We have experimented with a large variety of architectures, and while many simpler networks would have yielded unsatisfactory results, we designed different network structures to gain the different hierarchical features. The main advantage of machine-learned features is removing the bottleneck of manual feature engineering and reducing the demand for computing and memory resources.

4. FluidsNet methodology

4.1. Architecture

The overall structure of FluidsNet is shown in Fig. 2. The network takes Lagrangian fluid simulation data as input and then computes local features for each particle. After being processed through the background grid partition, voxel feature learning layers are employed to generate a 4D background grid feature tensor, which is fed into global feature learning layers. Then FluidsNet concatenates features from earlier layers to later layers in order to allow the network to combine local and global information. After the intermediate layers, the velocity field of every fluid particle is predicated as output.

The main contributions of FluidsNet is the different network structures used for different hierarchical features learning.

4.2. Feature extraction

There are 7 features of fluid particles used in FluidsNet as inputs, which are 3-dimensional position [*x, y, z*], 3-dimensional velocity [*u, v, w*] and fluid/solid marking *s/f*. At first sight, taking the acceleration field as predictive output is a feasible solution. After all, acceleration is the basic quantity in physics for measuring force and motion. However, over the course of numerous training runs, we have noticed that there are some inherent problems in the SPH simulation data. As mentioned in Ladický et al. (2015), the data analysis shows that when interacting with obstacles, many particles require relatively large corrections to avoid compression, corresponding to acceleration exceeding 10**g**. Thus, we take the velocity field as predictive output instead.

4.3. Local feature learning

In the SPH approach (Monaghan, 1992), the physical properties of each fluid particle are obtained by weighted summation

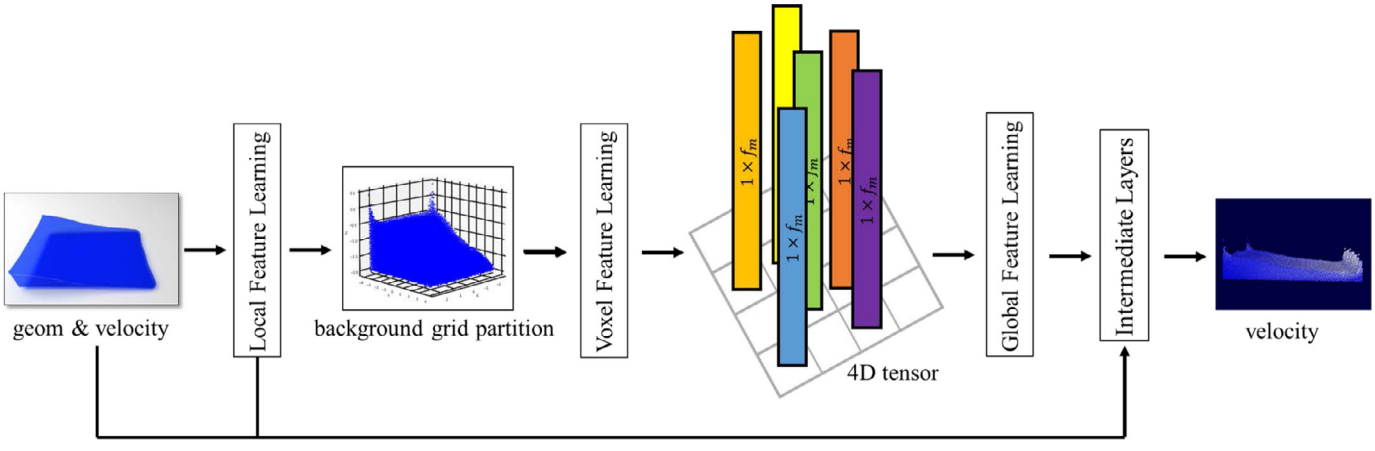


Fig. 2. Network architecture for our FluidsNet.

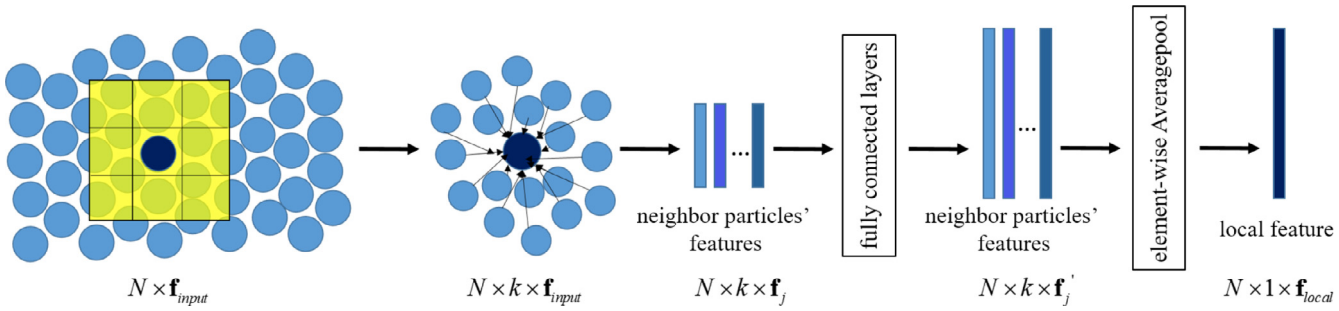


Fig. 3. Local feature learning layer.

of neighbor particles in its supporting domain. Correspondingly, we exploit local features for each particle by constructing a local neighborhood graph and applying convolution-like operations on neighbor particles, as shown in Fig. 3.

The local feature learning starts by finding k-nearest neighbors $\mathbf{p}_{j1}, \mathbf{p}_{j2}, \ldots, \mathbf{p}_{jk}$ for each particle $\mathbf{p}_i$ within its support domain. Please note that the number of neighbors k is not a fixed value, such as when the fluid particles are around the liquid-air interaction, the number of neighbors would be relatively small. Denote $\mathbf{f}_{input} = [x, y, z, u, v, w, s/f]$ for simplicity. Then calculate the relative distance between the neighbor particles and the particles $\Delta\mathbf{x} = \mathbf{x}_j - \mathbf{x}_i$. Replace the coordinate position $\mathbf{x}$ with relative position $\Delta\mathbf{x}$, and combined with other input features, we use $\mathbf{f}_j = [x_j - x_i, y_j - y_i, z_j - z_i, u, v, w, s/f]$ to represent the features of each neighbor particle at this time. Next, each $\mathbf{f}_j$ is transformed through the fully connected network (FCN) into a feature space, where we can aggregate information from the neighbor features $\mathbf{f}_j'$ to encode the local dynamic within the support domain. The FCN is composed of two linear layers with the hyperbolic tangent activation function. After obtaining particle-wise feature representations, we use element-wise AveragePooing across all $\mathbf{f}_j'$ associated with fluid particle $\mathbf{p}_i$ to get the local feature $\mathbf{f}_{local}$ for fluid particle $\mathbf{p}_i$.

In the local feature learning part, for each fluid particle, the input is a $k \times \mathbf{f}_j$ feature vector and output $\mathbf{f}_{local}$ is a vector in shape $1 \times f_{local}$, where f_{local} is determined by controlling the FCN. All fluid particle local features are encoded in the same way and they share the same set of parameters in FCN. Since the FCN is only calculated on every single particle, and the operation between particles is symmetric function AveragePooing, the output feature $\mathbf{f}_{local}$ could maintain invariance to the permutation of neighbor particles.

4.4. Voxel feature learning

Before voxel feature learning, we subdivide the simulation scene space R into equally 3-dimensional spaced background grid as shown in Fig. 2. Suppose the simulation space is in range W, D, H along the X, Y, Z axes. Define the background grid is of size g_w, g_d and g_h. The resulting gridded simulation scenes $\hat{R}$ is of size $\hat{W} = W/g_w$, $\hat{D} = D/g_d$, $\hat{H} = H/g_h$ accordingly, defined as $\hat{R} = [0, \hat{W}] \times [0, \hat{H}] \times [0, \hat{D}]$. Here, W, D, H are set to be a multiple of g_w, g_d and g_h for simplicity. Fluid particles are grouped according to the background grid they reside in.

For simplicity, Fig. 4 illustrates the voxel feature generation process for one 3D background grid. Without loss of generality, we use Voxel Feature Learning Layer-1 to describe the details in the following paragraph. Denote $\mathbf{G} = \{\mathbf{p}_i = [\mathbf{f}_{input}, \mathbf{f}_{local}], i \in [1, n]\}$ as a 3D background grid containing n fluid particles and $\mathbf{f}_1 = [\mathbf{f}_{input}, \mathbf{f}_{local}]$. Then, each $\mathbf{f}_1$ is transformed through a multi-layer perceptron network into a feature space, where we can aggregate information from the point features $\mathbf{f}_1'$ to encode the physical behavior within the voxel. Batch normalization (BN) is used for all MLP layers with

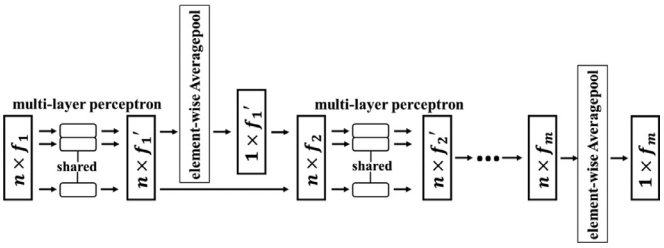


Fig. 4. Voxel feature learning layer.

a hyperbolic tangent activation function. After obtaining feature representations by particle, we use elementwise AveragePooling across all $\mathbf{f}_1'$ associated to $\mathbf{G}$ to get the locally aggregated feature $\mathbf{f}_G$ of size $1 \times f_1'$ for the current background grid. After computing the locally aggregated feature, we feed it back to per particle features by concatenating the locally aggregated feature with each of the particle features, as $\mathbf{f}_2 = [\mathbf{f}_1', \mathbf{f}_G]$. Then we extract new per particle features based on the combined particle features after several feature learning layers, the per particle feature is aware of information at different scales.After learning the voxel features of the m layer, we obtain the feature set $\mathbf{G} = \{\mathbf{f}_m\}_{i=1\ldots n}$ of size $n \times f_m$. The voxel feature is obtained by transforming the output feature set into a $1 \times f_m$ tensor via element-wise AveragePooling. Denote $\hat{C} = 1 \times f_m$ for simplicity.

4.5. Global feature learning

After the feature learning network, we obtain a list of voxel features, each uniquely associated to the spatial coordinates of a 3D background grid. For those empty grids, we use $\hat{C}$ random numbers that obey normal distribution $N(0, 10^{-2})$ to express the voxel features. Do not directly set the features of the empty grid to zero, which would lead to inaccurate output prediction. The obtained list of voxel-wise features can be represented as a full rank 4D tensor, of size $\hat{W} \times \hat{D} \times \hat{H} \times \hat{C}$ as shown in Fig. 2.

The global feature learning layers consist of a stack 3D convolution layers to obtain multi-resolution features and learn the behavior of fluid. The convolutional layers aggregate voxel-wise features within a progressively expanding receptive field, adding more context to the dynamic description. All outputs from each layer are activated using the hyperbolic tangent function. Inspired by Iandola et al. (2016), the 3D convolution layers are in a kernel of size (2,2,2), (3,3,3) or (3,2,3) with channel $\hat{C}$ to reduce the number of parameters and memory requirement. The generated feature map is in shape $1 \times 1 \times 1 \times \hat{C}$ and then reshaped to a $1 \times \hat{C}$ tensor. We apply two 1×1 convolutional layers to reduce the dimension of feature to $\hat{C}/4$. These 1×1 convolutions can be thought as strictly linear coordinate-dependent transformations in the filter space. Denote $\mathbf{f}_{global}$ as the global feature. Note that the final dimension of the global feature cannot be much larger than the

dimension of the local feature, in case the global features account for too much weight in predicting the velocity field and result in the homoplasy in all particles.

4.6. Intermediate layers

After learning the global feature, we feed it back to per particle features by concatenating the global feature with each of the particle input features and local features, as $\mathbf{p}_i = [\mathbf{f}_{input}, \mathbf{f}_{local}, \mathbf{f}_{global}]$. To predict the velocity field, the features are fed into four fully connected layers.

Use the hyperbolic tangent function to activate all outputs of each layer except the last layer. The last layer outputs 3D velocity field of each fluid particle. By using this velocity, the position of each particle is integrated as:

$$\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_{i\_prediction},$$

where Δt is the time step between two frames. The output of every fluid particle is encoded by $\mathbf{p}_{input} = [x, y, z, u, v, w, s/f]$, which will be used as input to the FluidsNet for prediction of the next time step.

5. Experiments and analysis

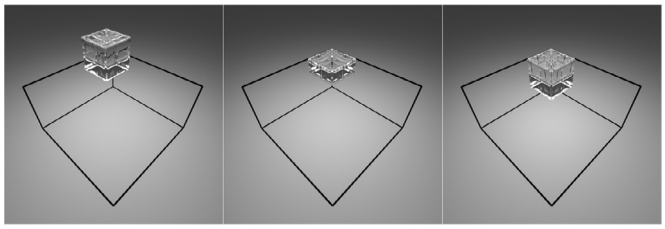
5.1. Data generation

For most deep learning approaches, it is crucial to have enough labeled training data sets. In this paper, we can generate enough ground truth data through simulation system. The training data is from IISPH algorithm (Ihmsen, Cornelis, Solenthaler, Horvath, & Teschner, 2014) evaluated on the randomized initial conditions. We have also taken other simulation algorithms such like WCSPH (Müller, Charypar, & Gross, 2003) and PCISH (Solenthaler & Pajarola, 2009). The results find the differences are not notable. We have tested our FluidsNet with three different 3D data sets, a cuboid dropping on a basin, a dam breaking and a liquid drop falling into a pool. These scenes covered a wide range of phenomena, from strong splashes to smooth surface waves, as well as falling and colliding of liquid. Examples of initial states can be found in Fig. 5. For each raw data set, we have simulated around 25 animations with 1500 frames of output per animation. Each of animation has fluid volumes of randomized size and position.

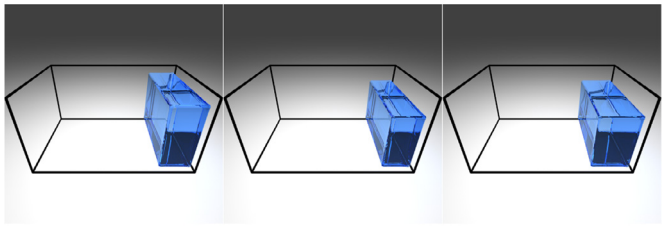
We applied uniform sampling for solid boundaries (such as the basin) and input them into the data set along with the fluid particles, distinguish them by feature *s|f*. Although solid particles do not move in our simulations, they play a decisive role in the model's understanding of fluid-solid interactions.

5.2. Experiment setup

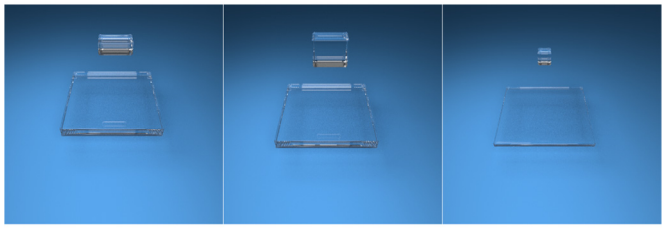
For training and running the trained network, we use Nvidia GeForce GTX 1080 Ti GPUs (each with 12GB Ram) and Intel Xeon E5-2623 CPUs. We summarize the detailed parameters of FluidsNet in Table 1. The scenes of a cuboid dropping on a basin and a liquid drop falling into a pool are the same size. Parameters of the network are identical for these two data sets. To clearly specify our network, we use the following notations. Let *FCN*(input, output) denotes a fully-connected layer; *conv2D*(input-channels, output-channels, (kernel size)) and *conv3D*(channels, (kernel size)) denotes a convolutional layer. Since the number of input and output channels of *conv3D* are the same in our network, we use channels to present both of them. Note that those operations without parameters are not included in Table 1, like element-wise AveragePooling and expanding. Our network does not require any additional regularization such as dropout or weight decay.



(a)The data set of a cuboid dropping on a basin.



(b)The dam breaking data set.



(c)The data set of liquid a drop falling into a pool.

Fig. 5. Examples of training data, obtained using IISPH solver.

5.3. Training

We use the same modalities for all training runs. We implement our FluidsNet with TensorFlow framework [2015] using its ADAM optimizer (Kingma & Ba, 2014) with an initial learning rate of 0.003 that decays to $\frac{1}{20}$th for second half of the training iterations. Full details are given in Table 2.

Since the simulation scenes are typically smooth and regular before 300 frames and after 1500 frames. So we target the scenes with high complexity, only 300 to 1500 frames per animation are suitable for training. We uniformly sample the raw data set, taking about 1 frame every 25 frames. 80% of the data set is used for training and the remaining 20% is used for testing. The number of training iterations is typically around 2.4k with 8 samples per batch. When the data sets were fully used, we randomly shuffled the samples and continue training.

The graphs in Fig. 6 illustrate the progress of the learning phase in our experiments. We observe that our FluidsNet is very stable and the training on each data set converges to a stable solution. This value is higher in the case of dam breaking. This is because stronger splashes from fluid-solid coupling can lead to higher complexity. The dam breaking examples require more training iterations (up to 161 minutes). The other two scenes finish the training process around 110 minutes.

5.4. Three-dimensional results

In the section, we show FluidsNet can reliably predict and synthesize dynamic flow fields in different data sets. We will focus on the details of performance, demonstrating that the plausibility of the overall flow can be preserved even with the strong splashes in the scene. Corresponding animation can be found in the sup-

Table 1
Network parameters and experiment settings of FluidsNet Architecture.

	A cuboid dropping on a basin & A liquid drop falling into a pool	Dam breaking
Input shape [N,7] Output shape [N,7]	FCN(7,7)	FCN(7,7)
	FCN(7,9)	FCN(7,9)
	Concatenate input and local features	
	Background grid patition	
For each grid: Input shape [n,16] Output shape [n,128]	conv2D(16,32,(1,1))	conv2D(16,32,(1,1))
	conv2D(32,64,(1,1))	conv2D(32,64,(1,1))
	conv2D(64,128,(1,1))	conv2D(64,128,(1,1))
4D tensor shape	[21,21,21,128]	[18,12,18,128]
Output shape [1,1,1,128]	conv3D(128,(2,2,2))	conv3D(128,(2,2,2))
	conv3D(128,(2,2,2))	conv3D(128,(3,2,3))
	conv3D(128,(2,2,2))	conv3D(128,(3,3,3))
	conv3D(128,(3,3,3))	
	Reshape[1,1,128]	
Input shape [1,1,128] Output shape [1,1,32]	conv2D(128,64,(1,1))	conv2D(128,64,(1,1))
	conv2D(64,32,(1,1))	conv2D(64,32,(1,1))
	Concatenate input, local and global features	
Input shape [N,48] Output shape [N,3]	FCN(48,32)	FCN(48,32)
	FCN(32,16)	FCN(32,16)
	FCN(16,8)	FCN(16,8)
	FCN(8,3)	FCN(8,3)

Table 2
Details of training runs.

	A Cuboid Dropping on a Basin (Fig. 7)	Dam breaking (Fig. 8)	Liquid Drop Falling into a pool (Fig. 9)
Raw data	1680 frames from 28 simulations	1440 frames from 24 simulations	1680 frames from 28 simulations
Training data set	1344	1152	1344
Testing data set	336	288	336
scene sizes	$7 \times 7 \times 7$ m	$6 \times 4 \times 6$ m	$7 \times 7 \times 7$ m
Batch size	8	8	8
Training epochs	2200	2600	2200
Model weights	9683	9811	9683
Training time(min)	107	161	110

plemental video. For each scene, we predict the velocity field and render the result with Blender.

Fig. 7 shows rendered results of a cuboid dropping on a basin, for which the scene size is $7 \times 7 \times 7$ m. The top row of Fig. 7 shows the 4th, 360th, 412th, 428th and 480th frames of the simulated animation, while the bottom row compares (left) FluidsNet predictions with (right) IISPH fluid simulations side-by-side in the selected areas of three frames. As shown in the picture, flow structure is plausibly reconstructed. In addition, The FluidsNet output can demonstrate small scale features close to the ground truth reference. More frame-by-frame comparisons can be seen in the accompanying video.

We also applied our trained 3D model to two other data sets, dam breaking and a liquid drop falling into a pool. The scene sizes are $7 \times 7 \times 7$ m and $6 \times 4 \times 6$ m. The rendered results are shown in Figs. 8 and 9. The results show that FluidsNet works reasonably in various fluid phenomena with random initial conditions. The top row of Fig. 8 shows the 104th, 316th, 420th, 516th and 632th frames of the simulated animation. As shown in the bottom row of Fig. 8, the bulk of the liquid dynamics are accurately predicted by our method, although some small scale errors would accumulate and lead to splash formations that deviate from the reference simulation.

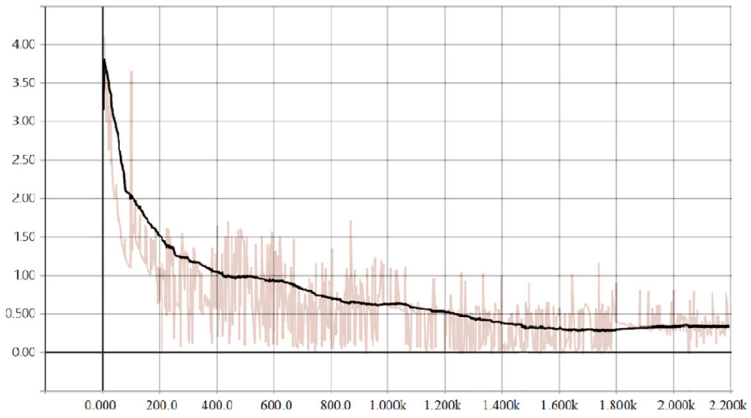
We have hidden the pool in the Fig. 9, in order to clearly show the flow detail of the fluid. The top row of Fig. 9 shows the 316th, 340th, 460th, 832th and 1200th frames of the simulated animation. As shown in the bottom row of Fig. 9, this approach works reliably for liquid-liquid interactions even in scenes with strong splashes.

5.5. Overall performance

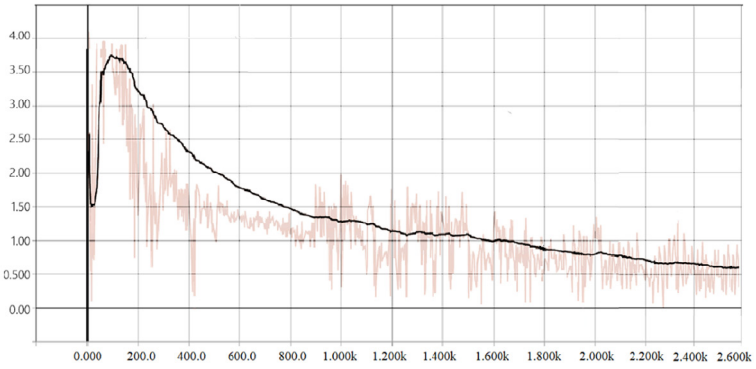
As for the hardware performance, we first compare the computation efficiency of FluidsNet with IISPH solvers. We run 50 forward-backward iterations for each case, including all CPU-GPU communications, and then calculate the average time per iteration. The statistics of time cost are summarized in Table 3. To facilitate comparisons with traditional simulation methods, we do not include the training time of FluidsNet when computing the maximum speedup, as pre-computation time is customarily reported separately. Instead, we included them in the previous discussion on training time.

In contrast to traditional solvers, FluidNet is capable of generating multiple frames independently and simultaneously. Therefore, we can effectively apply the network to the GPU batch, which outputs multiple velocity fields at once. The maximum batch size depends on the size of the network and the memory capacity of the hardware. The network evaluation time is inversely proportional to the maximum batch size supported by the GPU.

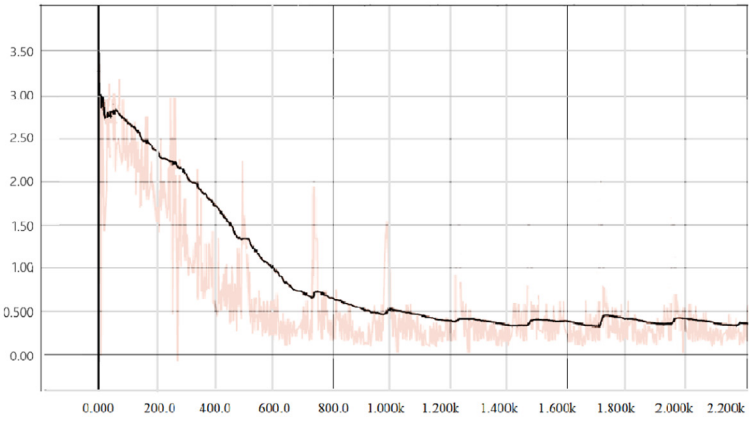
One goal of our approach is to make the model have considerable performance on speed-up large scene simulations. In addition to predicting similar scenes, the trained model is capable of predicting the scaled up scenes. As long as the size ratio of the large scene to the small scene is the same (for example, the aspect ratio of 1:1:1), the trained model from a small scene could be used to predicate the output of a large scene by modifying the voxel size proportionally. The left sub-figure of Fig. 10 shows the predicted results of our model with the same scene size as the training data set, while the middle and right columns show the predicted results with scenes that are magnified 1.5 $\times$ and 2 $\times$ in the same



(a) Training iterations on the data set of a cuboid dropping on a basin.



(b) Training iterations on the dam breaking data set.



(c) Training iterations on the data set of a liquid drop falling into a pool.

Fig. 6. Loss during training for three data sets.

Table 3
Comparisons on computation efficiency of single frame.

Scene	Particle Number	Simulation Time (s)	Precomputation Time (s) [Batch]	Prediction Time (s) [Batch]	Speed Up (×)
A cuboid dropping on a basin (Fig. 7)	28,623	1.58	3.42 [8]	0.58 [8]	21.79
Dam breaking (Fig. 8)	46,430	2.68	4.23 [8]	0.68 [8]	31.53
A liquid drop falling into a pool (Fig. 9)	66,987	5.13	6.83 [8]	0.83 [8]	49.44

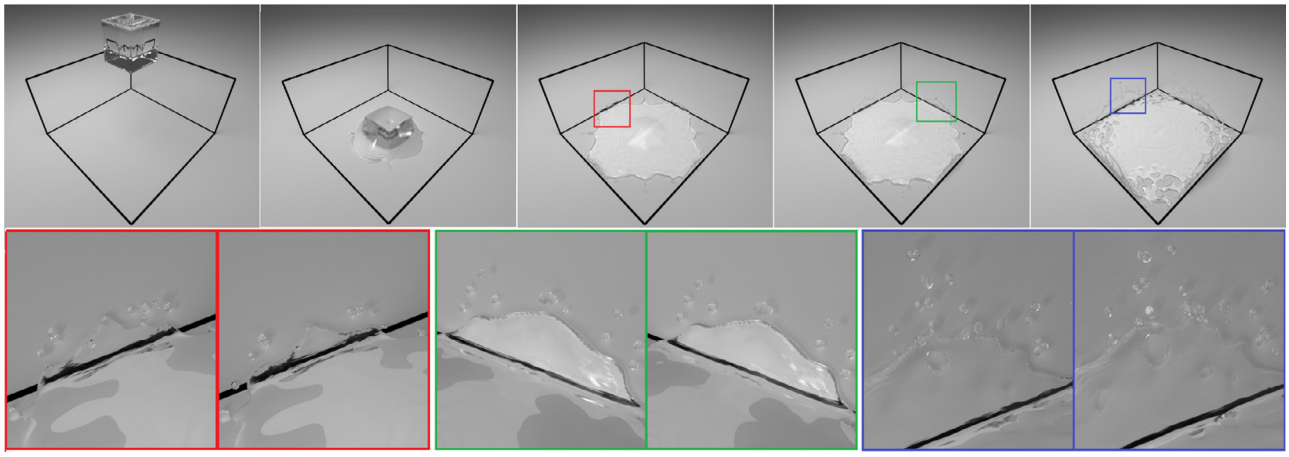


Fig. 7. Example frames of a cuboid dropping on a basin with FluidsNet. The bottom row compares (left) FluidsNet with (right) traditional fluid simulation side-by-side in the selected areas of three frames.

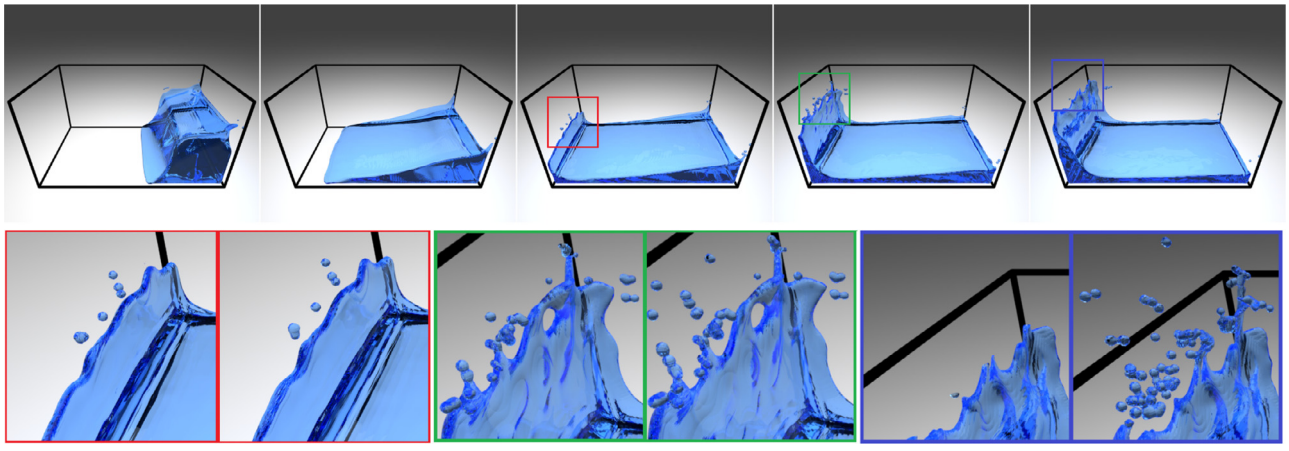


Fig. 8. Example frames of dam simulations with FluidsNet. The bottom row compares (left) FluidsNet with (right) traditional fluid simulation side-by-side in the selected areas of three frames.

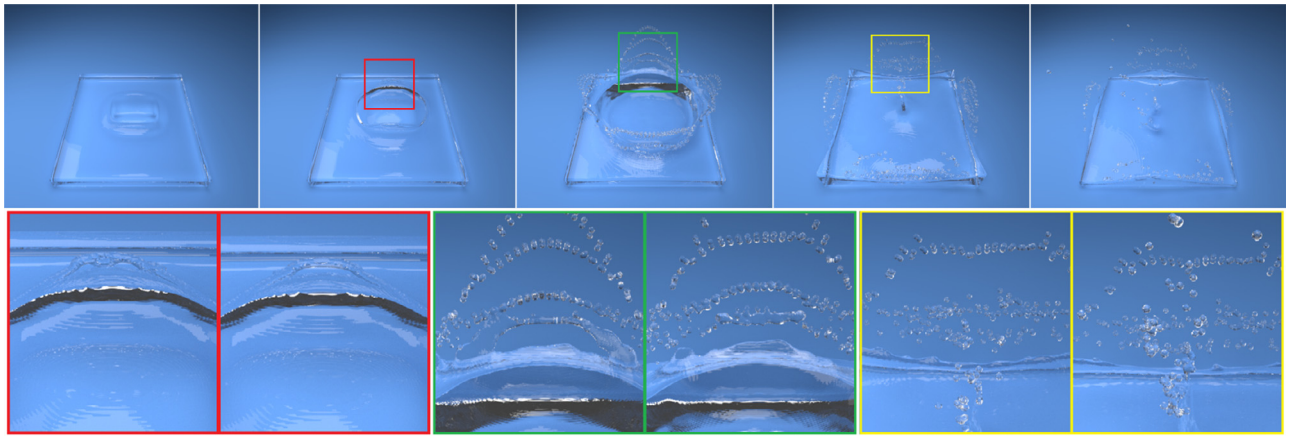


Fig. 9. Example frames of simulations of a cuboid dropping on a pool with FluidsNet. The bottom row compares (left) FluidsNet with (right) traditional fluid simulation side-by-side in the selected areas of three frames.

scale. The results show that this works reliably although the large-scale scenes are not included in the training set. Compared to the baseline simulation method, our model has great potential in the computational efficiency of large scene prediction. The corresponding measurements can be found in Table 4. More frame-by-frame details can be seen in the accompanying video.

Overall, from the comparison results, we see FluidNet significantly speeds up simulation time compared to regular pressure solvers. In particular, it pays off when targeting larger volumes. The pre-computation part of the network is time consuming, but considering the ability to run multiple samples in batches, this approach still has much better performance.

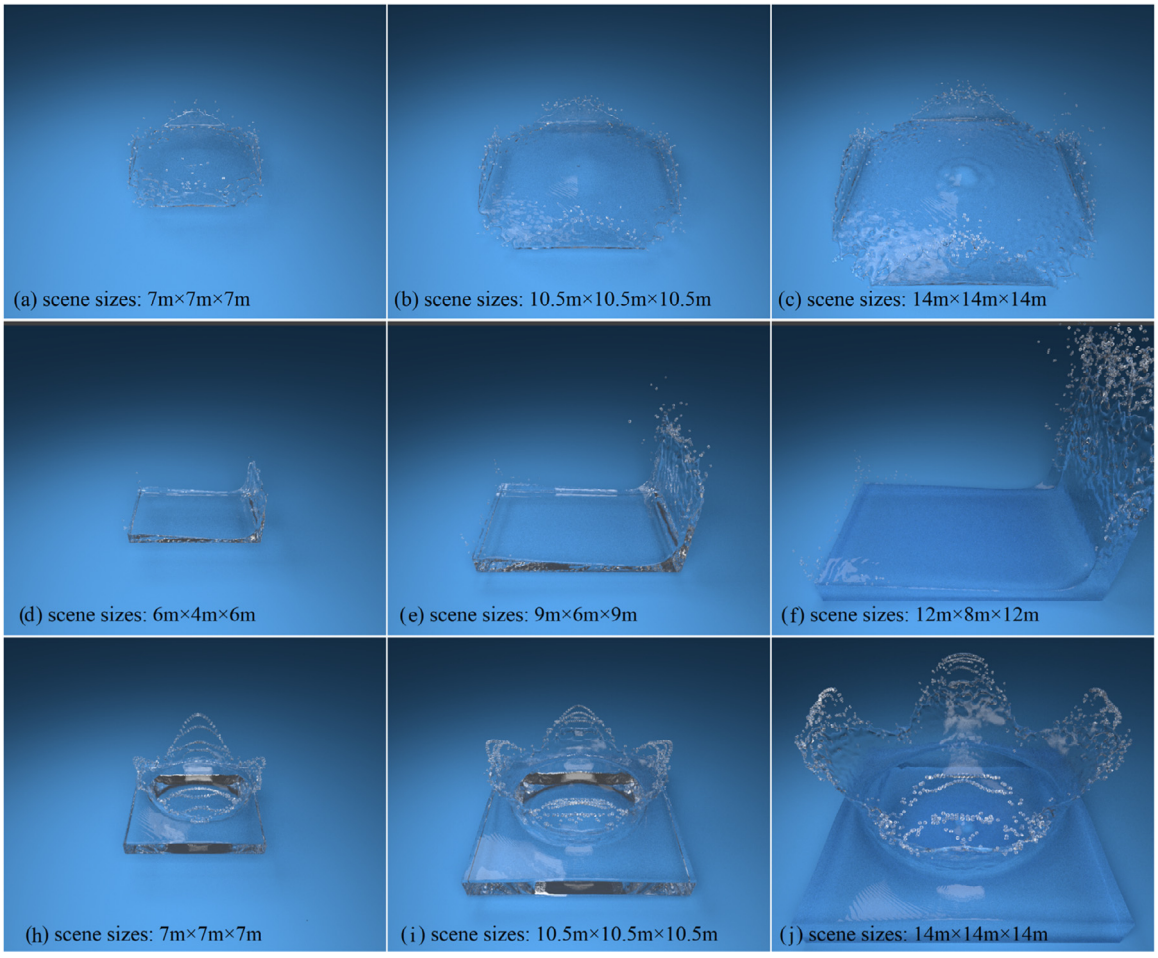


Fig. 10. The predicted large scene results obtained by using the training model from the corresponding small scenes.

Table 4
Performance measurement of predicting large scenes.

Scene	Scene Size (m^3)	Particle Number	Simulation Time (s)	Prediction Time(s)	Speed Up (×)
Fig. 10b	10.5 × 10.5 × 10.5	73,234	4.06	0.75	5.41
Fig. 10c	14 × 14 × 14	127,417	7.47	1.07	6.98
Fig. 10e	9 × 6 × 9	139,788	8.33	1.11	7.50
Fig. 10f	12 × 8 × 12	305,460	23.51	1.78	13.21
Fig. 10i	10.5 × 10.5 × 10.5	270,272	17.43	1.70	10.25
Fig. 10j	14 × 14 × 14	618,043	50.51	2.76	18.30

5.6. Impact of loss function and global feature

Several factors can affect the prediction accuracy of our model. We evaluated different physical quantities as baseline for predictions. In particular, we experimented with acceleration field as predictive output. The result of L2 error is shown in Fig. 11. As fluid particles would have relatively large acceleration to avoid compression when interacting with obstacles, it is difficult for the model to learn reasonable dynamics.

We also evaluate the velocity prediction of the model without global feature. A comparison between FluidsNet model and the model without global feature is shown in Fig. 12. From left to right, the images show the velocity field plot in random initial state, in iteration and convergence state. The larger the velocity field is, the whiter the fluid particle color will be. When the velocity approaches to 0, the fluid particles are visualized as blue. The experimental results show that the model without global feature would bring the velocity field near 0 iteratively.

6. Discussion and limitations

Although FluidNet can provide reasonable predictions and robust simulations for Lagrangian fluid simulation, it contains several limitations. We have other numerous interesting extensions for this work. Besides the basic fluid behaviors, interactions between fluids and moving obstacles, long-term prediction, and non-Newton fluid simulation would be new challenges.

In addition, it would also be helpful to explore the use of other data structures for global feature learning. In this work, we subdivide the simulation scene space equally into 3D spaced background grid to gain regular data formats for convolutional operation. Some other data structures, such as k-d trees or octrees, are also worth trying.

Finally, similar to the weakness of all machine learning approaches, FluidNet needs to be trained for the physical effects in each data set. It is sensitive to the scene size ratio. Except for this,

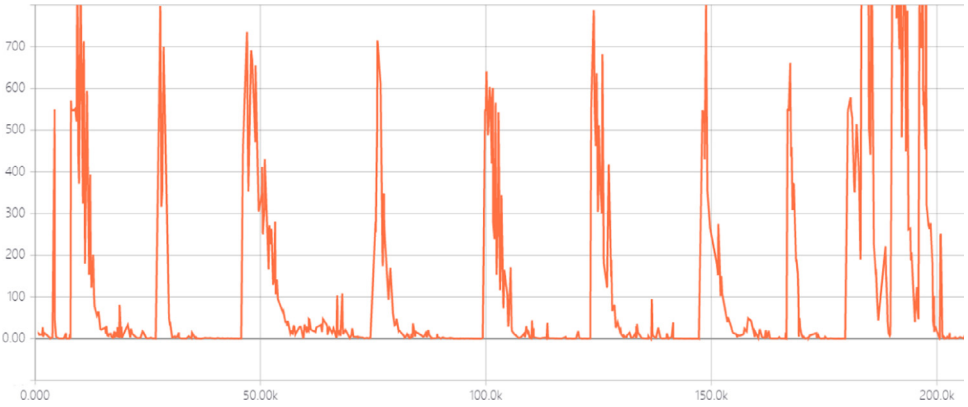


Fig. 11. L2 error plot when taking the acceleration field as predictive output.

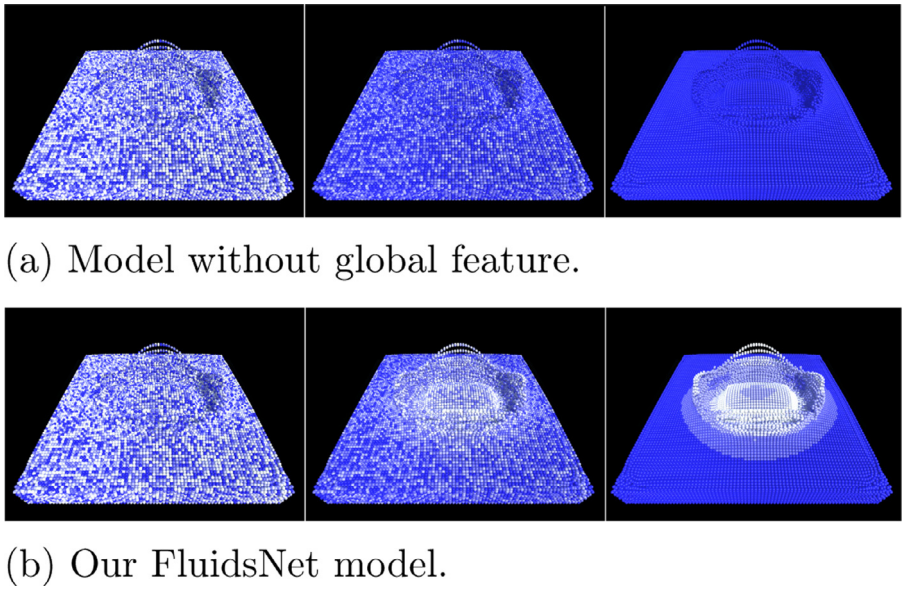


Fig. 12. Velocity field visualization results of a liquid drop falling into a pool example (from left to right: random initial state, in iteration, convergence state).

our model still has considerable performance on scenario reuse and speed-up large scene simulations.

7. Conclusions

In this paper we have presented an intelligent system to learn fluid dynamics from Lagrangian fluid simulation data. We deal with irregular Lagrangian data structures by processing every fluid particle identically and independently and capturing space structure and interactions among particles with symmetric functions. This Network architecture is carefully designed to achieve high quality fluid simulations with challenging details, which is why we used different network structures to gain the different hierarchical features. Experiment results show that FluidsNet is sufficiently accurate and stable to predict velocity field for various simulation scenes with random initial states. In addition, our method leads to significant speed-ups compared to previous method and has considerable performance on speed-up large scene simulations.

We believe that FluidsNet represents an important step towards intelligent system powered simulation algorithms with irregular data structures. We have demonstrated that intelligent system is able to infer velocity field and Navier-Stokes equations, which could encourage other researchers to investigate other physical learning problems. Also, since the simulation system can generate enough ground truth data for training, research in this field may contribute to the development of intelligent system.

In the future work, we plan to combine learning approach with other data structures such as k-d trees to reduce the requirement for memory and computing resources. Another interesting direction is to extend intelligent system with temporal learning to obtain long-term predictions.

There is much work could be extended on the current state of knowledge in this field. E.g., the research can be further extended to non-Newtonian fluids or multiphase flows with complex physical characteristics, using data obtained from sensors at industrial production sites to correct and predict fluid animations generated by intelligent systems.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No. 61873299, No. 61702036, No. 61902022) and The National Key Research, Development Program of China (Grant No. 2016YFB0700502, 2016YFB1001404).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.eswa.2020.113410.

References

Agrawal, P., Nair, A. V., Abbeel, P., Malik, J., & Levine, S. (2016). Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in neural information processing systems* (pp. 5074–5082).

Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al. (2016). Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems* (pp. 4502–4510).

Battaglia, P. W., Hamrick, J. B., & Tenenbaum, J. B. (2013). Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 201306572.

Chang, M. B., Ullman, T., Torralba, A., & Tenenbaum, J. B. (2016). A compositional object-based approach to learning physical dynamics. arXiv:1612.00341.

Bonev, B., Prantl, L., & Thuerey, N. (2017). *Pre-computed liquid spaces with generative neural networks and optical flow.*. arXiv:1704.07854.

Chu, M., & Thuerey, N. (2017). Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics (TOG), 36*(4), 69.

Cressie, N., & Wikle, C. K. (2015). *Statistics for spatio-temporal data*. John Wiley & Sons.

Denil, M., Agrawal, P., Kulkarni, T. D., Erez, T., Battaglia, P., & de Freitas, N. (2016). Learning to perform physics experiments via deep reinforcement learning. arXiv:1611.01843.

Ehrhardt, S., Monszpart, A., Mitra, N. J., & Vedaldi, A. (2017). Learning a physical long-term predictor. arXiv:1703.00247.

Higo, E., Okada, N., Hipel, K. W., & Fang, L. (2017). Cooperative survival principles for underground flooding: Vitae system based multi-agent simulation. *Expert Systems with Applications, 83*, 379–395.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. arXiv:1602.07360.

Ihmsen, M., Cornelis, J., Solenthaler, B., Horvath, C., & Teschner, M. (2014). Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics, 20*(3), 426–435.

Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., & Solenthaler, B. (2018). Deep fluids: A generative network for parameterized fluid simulations. arXiv:1806.02071.

Kim, S., Son, Y.-J., Tian, Y., Chiu, Y.-C., & Yang, C. D. (2017). Cognition-based hierarchical en route planning for multi-agent traffic simulation. *Expert Systems with Applications, 85*, 335–347.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

Klokov, R., & Lempitsky, V. (2017). Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Computer vision (ICCV), 2017 IEEE international conference on* (pp. 863–872).

Ladický, L., Jeong, S. H., Solenthaler, B., Pollefeys, M., & Gross, M. (2015). Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics, 34*(6), 1–9.

Lerer, A., Gross, S., & Fergus, R. (2016). Learning physical intuition of block towers by example. arXiv:1603.01312.

Li, W., Leonardis, A., & Fritz, M. (2016). Visual stability prediction and its application to manipulation. arXiv:1609.04861.

Li, Y., Bu, R., Sun, M., & Chen, B. (2018). PointCNN. arXiv:1801.07791.

Lucy, L. B. (1977). A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal, 82*, 1013–1024.

Ma, P., Tian, Y., Pan, Z., Ren, B., & Manocha, D. (2018). Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (TOG), 37*(4), 96.

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

Mottaghi, R., Bagherinezhad, H., Rastegari, M., & Farhadi, A. (2016). Newtonian scene understanding: Unfolding the dynamics of objects in static images. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3521–3529).

Müller, M., Charypar, D., & Gross, M. (2003). Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation* (pp. 154–159).

Peng, X. B., Abbeel, P., Levine, S., & van de Panne, M. (2018). DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. arXiv:1804.02717.

Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 1*(2), 4.

Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems* (pp. 5099–5108).

Solenthaler, B., & Pajarola, R. (2009). Predictive-corrective incompressible SPH. In *ACM transactions on graphics (TOG): Vol. 28* (p. 40).

Tompson, J., Schlachter, K., Sprechmann, P., & Perlin, K. (2016). Accelerating Eulerian fluid simulation with convolutional networks. arXiv:1607.03597.

Wagner, N., & Agrawal, V. (2014). An agent-based simulation system for concert venue crowd evacuation modeling in the presence of a fire disaster. *Expert Systems with Applications, 41*(6), 2807–2815.

Wang, P.-S., Liu, Y., Guo, Y.-X., Sun, C.-Y., & Tong, X. (2017). O-CNN: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG), 36*(4), 72.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., & Solomon, J. M. (2018). Dynamic graph CNN for learning on point clouds. arXiv:1801.07829.

Watters, N., Tacchetti, A., Weber, T., Pascanu, R., Battaglia, P., & Zoran, D. (2017). Visual interaction networks. arXiv:1706.01433.

Wiewel, S., Becher, M., & Thuerey, N. (2018). Latent-space physics: Towards learning the temporal evolution of fluid flow. arXiv:1802.10123.

Wilkie, D., Sewall, J., & Lin, M. (2013). Flow reconstruction for data-driven traffic animation. *ACM Transactions on Graphics (TOG), 32*(4), 89.

Xie, Y., Franz, E., Chu, M., & Thuerey, N. (2018). tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. arXiv:1801.09710.

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

Zhou, Y., & Tuzel, O. (2017). Voxelnet: End-to-end learning for point cloud based 3d object detection. arXiv:1711.06396.