Fast Fluid Simulation via Dynamic Multi-Scale Gridding

Jinxian Liu[1], Ye Chen[1], Bingbing Ni[1]*, Wei Ren[2], Zhenbo Yu[1], Xiaoyang Huang[1]

[1]Shanghai Jiao Tong University, Shanghai 200240, China
[2]Huawei Hisilicon
{liujinxian, chenye123, nibingbing}@sjtu.edu.cn
renwei.ai@huawei.com, {yuzhenbo, huangxiaoyang}@sjtu.edu.cn

Abstract

Recent works on learning-based frameworks for Lagrangian (*i.e.*, particle-based) fluid simulation, though bypassing iterative pressure projection via efficient convolution operators, are still time-consuming due to excessive amount of particles. To address this challenge, we propose a dynamic multi-scale gridding method to reduce the magnitude of elements that have to be processed, by observing repeated particle motion patterns within certain consistent regions. Specifically, we hierarchically generate multi-scale micelles in Euclidean space by grouping particles that share similar motion patterns/characteristics based on super-light motion and scale estimation modules. With little internal motion variation, each micelle is modeled as a single rigid body with convolution only applied to a single representative particle. In addition, a distance-based interpolation is conducted to propagate relative motion message among micelles. With our efficient design, the network produces high visual fidelity fluid simulations with the inference time to be only **4.24 ms/frame** (with 6K fluid particles), hence enables real-time human-computer interaction and animation. Experimental results on multiple datasets show that our work achieves great simulation acceleration with negligible prediction error increase.

Introduction

Simulations of complex physics are invaluable to many science and engineering disciplines, *e.g.*, computational fluid dynamics, computer graphics. A wide range of physical processes such as the motion of water-like fluids shows its great application value in animation and the human-computer interaction. These applications put forward high requirements for the efficiency of simulation systems.

As we know, most physical processes such as fluid mechanics are described by partial differential equations. Traditional fluid simulators usually design numerical solvers to solve the Navier-Stokes equations. However, such methods require substantial computational resources and are hard to extend. Thereby, many works (Li et al.; Ummenhofer et al.; Tompson et al.; Morton et al.; Battaglia et al.; Mrowca et al.; Sanchez-Gonzalez et al.; Kim et al.) seek learning-based methods for help. Training simulators directly from observed data is an attractive alternative to traditional simulators, due to that learning-based methods often show better generalization ability and are more extensible. There are some choices of fluid representations, among which Lagrangian representations based on particles are particularly popular and widely used. However, a fluid sample usually contains thousands of particles and even tens of thousands of particles when the scene is complicated. To build a high-precision and physical-level fluid simulation system, convolutions are usually performed on each particle to facilitate particle-wise motion prediction. Although some efficient convolution operators (Thomas et al.; Ummenhofer et al.) have been proposed, performing convolutions on such kinds of irregular data is still expensive, especially when the number of particles is large. We can sacrifice some precision and achieve a significant speedup in some application-level scenarios that only pursue visual fidelity simulation such as animation and the human-computer interaction.

To this end, we develop a dynamic multi-scale gridding method and modify the second-order Runge-Kutta for faster and high visual fidelity fluid simulation. The framework of our method is shown in Figure 1. During training, we generate multi-scale micelles by grouping particles that share similar motion patterns for training data based on the Octree generation algorithm. Specifically, we first partition all particles in Euclidean space into eight octants. Then we define a physical quantity named velocity-entropy to describe the chaos (*i.e.*, the degree of internal motion variation) of each octant. We iteratively partition the space into multi-scale octants/grids/micelles until the chaos of all octants is lower than a threshold and return the depth of octant to which each particle belongs. Through this procedure, a Lagrangian fluid sample is divided into multi-scale local areas and each area contains particles that share similar motion patterns. Next, we approximately take each micelle as a rigid body, and perform the efficient Continuous Convolution (Ummenhofer et al.) on each micelle by taking its gravity center as the center point and particles in this micelle as neighbors. Fast linear interpolation between gravity centers and fluid particles is applied to propagate message between micelle features and extract particle-wise features. Finally, we construct a lightweight ConvNet to learn fluid mechanics. However, computing velocity-entropy is time-consuming for inference. Hence we train the ConvNet to

*Corresponding Author: Bingbing Ni.

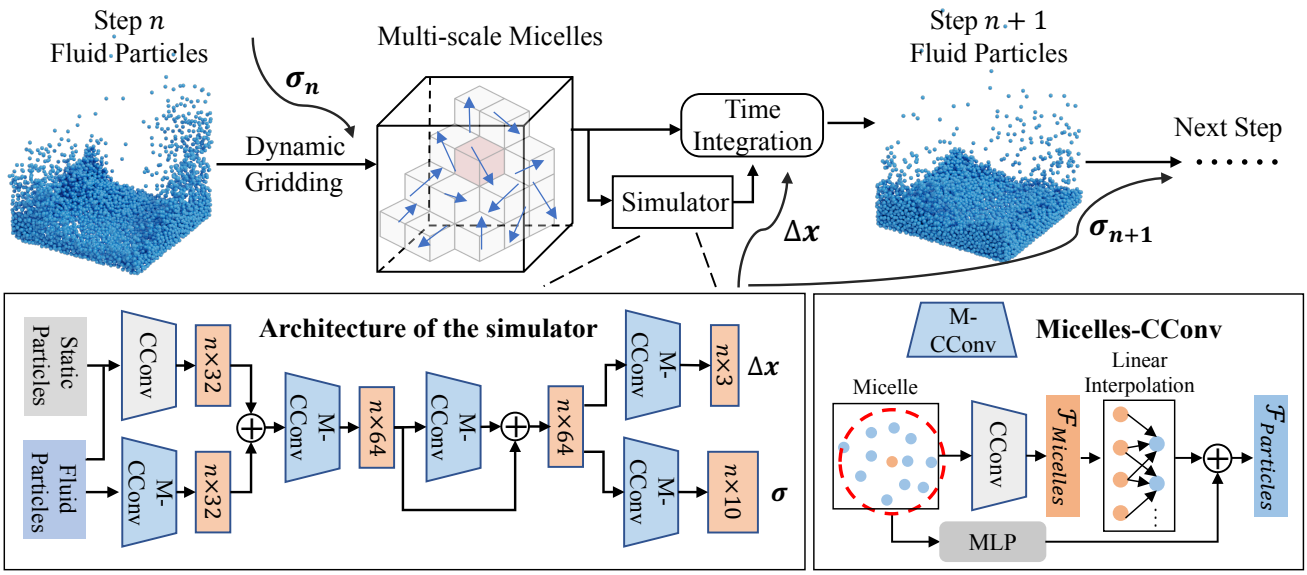

Figure 1: The overall framework of our method. CConv and M-CConv denote Continuous Convolution and our proposed Micelle-CConv respectively. Static denotes the static particles sampled from box. The number represents the channel length.

predict a scale factor supervised by the depth returned during multi-scale gridding for each particle. During inference, we efficiently generate multi-scale micelles directly according to the statistics of the predicted scale factors. To achieve further acceleration, we double the time-step and modify the second-order Runge-Kutta to be adaptive to learning-based method and improve accuracy during inference. Our method achieves high visual fidelity fluid simulation with extremely fast speed. Moreover, our proposed multi-scale gridding algorithm can be easily combined with most learning-based Lagrangian simulation systems and achieves acceleration.

Related Work

Conventional Fluid Simulation

Smoothed Particle Hydrodynamics (SPH) (Monaghan) is one popular particle-based method, which evaluates pressure and viscosity forces around each particle, and updates velocities and positions of particles accordingly. Many variants of SPH (Solenthaler and Pajarola; Bender and Koschier) have also been widely used to model complex real-world phenomena for visual effects. Position-based dynamics (PBD) (Müller et al.) and Material point method (MPM) (Sulsky, Zhou, and Schreyer) are methods designed for interacting, deformable materials. Incompressibility and collision dynamics involve resolving pairwise distance constraints between particles, and directly predicting their position changes in PBD. The hybrid Eulerian/Lagrangian MPM uses a continuum description of the governing equations and utilizes user-controllable elasto-plastic constitutive models. Recently, some differentiable particle-based simulators (Hu et al.; Holl, Thuerey, and Koltun) have been proposed to solve simulation and control problems. Although these conventional simulation systems are robust and mature, they require substantial computational resources.

Learning-based Fluid Simulation

Recently, numerous works (Ladicky et al.; Kim et al.; Li et al.; Ummenhofer et al.; Sanchez-Gonzalez et al.) have been proposed to learn fluid simulators using neural networks. They show that learning-based methods are capable of synthesizing plausible fluid simulations. Besides, computational efficiency improvement is the most attractive benefit for these families of methods, which enable fluid simulations to gain accelerations on the order of one or two compared with conventional simulators. A fast SPH simulation developed in (Ladicky et al.) depends on hand-crafted features and uses regression forests to regress motion vectors. It is not an end-to-end learning-based system. As for other learning-based fluid simulators, there are three main sets of methods, GAN-based, CNN-based and GNN-based methods, which utilize different kinds of deep learning techniques. Kim *et al.* (Kim et al.) propose a generative network to conduct fluid simulations, and it is reported to gain 700X performance speed-ups compared with conventional CPU solvers. Ummenhofer *et al.* (Ummenhofer et al.) present a simple, novel and effective N-D convolution to the continuous domain (*i.e.*, Continuous Convolution), which can be applied to perform particles-based fluid simulation. The ConvNet constructed in (Ummenhofer et al.) shows obviously lower error compared to previous learning-based methods such as DPI-Nets (Li et al.), which indicates its ability to capture better inherent physics. Moreover, the Continuous Convolution outperforms previous convolutions (Schenck and Fox; Wang et al.; Fey et al.; Thomas et al.) designed for point cloud learning and as reported it accelerates simulations several times. (Sanchez-Gonzalez et al.; Pfaff et al.) are GNN-Based methods, and they perform better in generalization and have more flexibilities. However, GNN-based methods are usually computational inefficient. Although these

learning-based methods are relatively more efficient than conventional methods, it is hard to apply these learning-based methods to real-time applications. In this work, we propose an efficient particle and learning based system to perform fast and high visual fidelity fluid simulation.

Preliminary: Fluid Simulation with Continuous Convolutions

Ummenhofer *et al.* (Ummenhofer et al.) present an efficient ConvNet based on Continuous Convolution (*i.e.*, CConv) for particle-based fluid simulation. We choose this architecture as the backbone of our method due to its efficiency, thereby we review it in this section. CConv performs better than more complicated representations (Schenck and Fox; Wang et al.; Fey et al.; Thomas et al.), despite its simplicity. With the help of simple linear interpolation, the grid-based filter representation usually used for discrete convolutions is extended to the continuous domain. It can assign varying filter values to points at arbitrary/continuous positions while retaining the compactness and efficiency, only by efficient lookup of a parameter grid.

A lightweight ConvNet is established by stacking CConvs to perform fluid simulation. The input fluid particle is represented as $(\mathbf{x}_i^n, [1, \mathbf{v}_i^n, \nu_i])$, where $\mathbf{x}_i^n$ denotes the position of particle p_i^n at time-step n, $\mathbf{v}_i^n$ and ν_i denote the corresponding velocity and viscosity. The intermediate positions $\mathbf{x}_i^{n*}$ and velocities $\mathbf{v}_i^{n*}$ are computed as:

$$\mathbf{v}_i^{n*} = \mathbf{v}_i^n + \Delta t \mathbf{a}_{ext}, \tag{1}$$

$$\mathbf{x}_i^{n*} = \mathbf{x}_i^n + \Delta t \frac{\mathbf{v}_i^n + \mathbf{v}_i^{n*}}{2}. \tag{2}$$

The vector $\mathbf{a}_{ext}$ represents the acceleration controlled by external forces such as gravity. Then the ConvNet is applied to pass message among fluid particles and handle collisions with the environment. The particles on the boundaries of the scene are sampled and treated as static particles s_j, whose normals $\mathbf{n}_j$ are taken as input features. The ConvNet takes both intermediate fluid particles and static particles as inputs, and predicts correction of the position which results from all particle interactions including the collision handling with the scene.

$$[\Delta\mathbf{x}_1, ..., \Delta\mathbf{x}_N] = ConvNet(\{p_1^{n*}, ..., p_N^{n*}\}, \{s_1, ..., s_M\}). \tag{3}$$

Given the position correction, positions and velocities for next time-step $n+1$ are updated as:

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^{n*} + \Delta\mathbf{x}_i, \tag{4}$$

$$\mathbf{v}_i^{n+1} = \frac{\mathbf{x}_i^{n+1} - \mathbf{x}_i^n}{\Delta t}. \tag{5}$$

Methodology

Dynamic Multi-Scale Gridding

CConv is a relatively efficient convolution operator designed for irregular particle sets, and its neighborhood searching module is based on a fully optimized spatial hashing on GPU. However, a real-world Lagrangian fluid sample usually contains an excessive amount of particles, thereby applying CConv for each particle is still a very time-consuming procedure. To achieve further acceleration and realize extremely rapid fluid simulations, we propose a *Dynamic Multi-Scale Gridding* algorithm to divide a large-scale fluid sample into multi-scale micelles according to their positions and velocities. We hold a point that particles share similar motion patterns could be approximately taken as a rigid body. Hence we group particles into micelles with different scales, and apply continuous convolutions for each micelle.

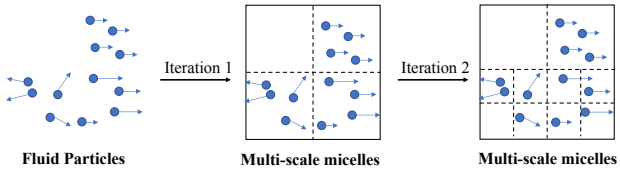

Figure 2: The process of multi-scale gridding. The input 2D fluid particles are first divided into 4 quadrants according to their positions. Then we compute the velocity-entropy for each quadrant. The velocity of the particle is denoted by blue arrows in the figure, whose direction and length indicate the direction and magnitude of the velocity respectively.

Our dynamic multi-scale gridding algorithm is based on the Octree data structure. Specifically, we first partition the space by subdividing it into eight octants. Then we compute the velocity-entropy of particles for each octant. The velocity-entropy is used to measure the internal motion variation and it is calculated by computing average L_2 distance of velocity vectors in pairs, which is formulated as:

$$e = \frac{\sum_{i=1,j=1,i\neq j}^{N_h} ||\mathbf{v}_i^{n*} - \mathbf{v}_j^{n*}||_2}{N_h \times (N_h - 1)}. \tag{6}$$

N_h denotes the number of particles in the octant h. The total number of divided octants is denoted by H. e implies the velocity difference of particles in an octant, *i.e.*, the degree of relative motion among particles. There is obvious relative motion when particles move in the same direction with different velocity magnitudes or in different directions with the same velocity magnitude. If the velocity-entropy e is larger than the threshold δ we set, we divide this octant into a more fine-grained scale. The above process is recursively performed until the velocity-entropy of all octants in different scales are smaller than δ or the maximal depth of Octree is larger than our setting. As a result, areas with large internal motion variation are divided into small scale micelles. We present a 2D example in Figure 2 to show the physical meaning of e and overview of multi-scale gridding.

After generating multi-scale micelles, we treat each micelle as a rigid body and set its gravity point $\mathbf{g}_h^{n*}$ as the center of all particles in this micelle. To generate motion correction for each fluid particle, we also use the similar ConvNet as (Ummenhofer et al.). However, we only perform CConv on each micelle rather than each particle to achieve faster simulation speed. Note that the number of micelles is related to

motion patterns of particles and the threshold δ. In our experiments, the number of particles is about 10-20 times as micelles. For each micelle, we compute its features by applying CConv for its gravity point and taking particles in this micelle as its neighbors. So we do not need nearest neighbor search (NNS) to gather neighbors anymore. Moreover, the interaction among micelles is also a key point for generating features and predicting the motions of particles. To achieve this, we use simple and efficient linear interpolation to generate particle-wise features according to micelle features, positions of gravity points and particles. We also directly process the features of each particle via a fully-connected stream, and merge them with interpolated particle-wise features. We name this operator as Micelle-CConv, which is shown in the lower right of Figure 1.

Learning Scale Factor for Fast Gridding

Although we greatly reduce the number of convolution operations by changing the convolution object from particles to micelles, the computation of velocity-entropy is time-consuming due to its high complexity. To address this problem, we propose a fast version of multi-scale gridding used for inference. We first generate multi-scale micelles for all training data using the original algorithm presented in previous section, and return Octree depth σ_i^n of each particle. The larger depth σ_i^n indicates that the particle p_i^n belongs to a micelle with a smaller scale. Hence we take σ_i^n as a factor that indicates which scale the particle p_i^n belongs to. Then we use the ConvNet to predict the scale factor $\widetilde{\sigma}_i^{n+1}$ of each particle at next time-step, which is supervised by generated ground-truth σ_i^{n+1}. The ConvNet is based on our proposed Micelle-CConv and shares a similar structure with (Ummenhofer et al.). The predictions of position correction and scale factors share the same backbone, *i.e.*, the backbone followed by two parallel heads to predict position correction and scale factors respectively. The overall framework and the architecture of the simulator are shown in Figure 1. The scale factors prediction is formulated as a classification task, and the number of classes is set to the maximal depth of the Octree.

During inference, we generate multi-scale micelles for the next frame according to the statistics of predicted scale factors. Instead of computing velocity-entropy for each micelle, we only calculate the mean and variance of the predicted scale factors for each micelle. Then we decide whether to continue dividing each micelle according to the mean and variance of predicted scale factors. The specific procedure is shown in Algorithm 1.

Algorithm 1: Dynamic Multi-Scale Gridding (Inference version).

Input: A Particle Set $P = \{p_1^n, p_2^n, ..., p_N^n\}$ with corresponding positions and scale factors.
Output: A multi-scale micelles set.

Initialization: root=$\{P, center, depth = 0\}$, root is a leafnode without children initially;
for p_i^n in P **do**
  Find corresponding leafnode according to coordinates;
  data_size ← # particles of current node;
  current depth ← # edges from current node to the root;
  if *data_size <16 or current depth = maximal depth* **then**
    Put p_i^n into current node;
  else if *data_size >100* **then**
    Subdivide current node into 8 octants;
    for *each particle in current node* **do**
      Put particle into corresponding octant according to coordinates;
    Put p_i^n into corresponding octant according to coordinates;
  else
    Put p_i^n into current node;
    mean ← Mean (the mean of the scale factors of all the particles in current node);
    var ← Variance (the variance of the scale factors of all the particles in current node);
    if *(var >ε) or (mean >current depth)* **then**
      Subdivide current node into 8 octants;
      for each particle in current node **do**
        Put particle into corresponding octant according to coordinates;
    else
      Continue;

The NN-RK2 Time Integration for Faster Inference

An intuitive way to further accelerate simulation is to increase the step size. For example, we can increase the time-step to $2\Delta t$ and train the network to predict $\mathbf{x}_i^{n+2}$, $\mathbf{v}_i^{n+2}$ by taking $\mathbf{x}_i^n$, $\mathbf{v}_i^n$ as inputs. Then we can get the intermediate results $\mathbf{x}_i^{n+1}$ and $\mathbf{v}_i^{n+1}$ by interpolation. This process is about twice as fast as the original version. However, it will result in a very large prediction error when we directly predict $\mathbf{x}_i^{n+2}$, $\mathbf{v}_i^{n+2}$ with increased time-step (as shown in Table 3). Actually, it is Euler Method with doubled time-step, hence we can ask high-order Runge-Kutta (*i.e.*, a family of high-precision numerical solutions for Differential Equations) for help to alleviate this problem. Motivated by the second-order Runge-Kutta (RK2) with two stage, we modify it to be adaptive to our learning-based method and improve prediction accuracy when we set the time-step to $2\Delta t$. We call it NN-RK2 and formulate this process as follows:

$$\mathbf{x}_i^{n+2} = \mathbf{x}_i^n + 2\Delta t K_2, \tag{7}$$

$$K_1 = f(n, \mathbf{x}_i^n) = \mathbf{v}_i^n, \tag{8}$$

$$K_2 = f(n+1, \mathbf{x}_i^n + \Delta t K_1) = \mathbf{v}_i^{n+1}. \tag{9}$$

Hence we can simplify above formulations as:

$$\mathbf{x}_i^{n+2} = \mathbf{x}_i^n + 2\Delta t \mathbf{v}_i^{n+1}. \tag{10}$$

Given the input $\mathbf{x}_i^n$ and $\mathbf{v}_i^n$, we use the ConvNet to predict $\mathbf{x}_i^{n+1}$ and $\mathbf{v}_i^{n+1}$. Then we directly use the Eq. 10 to get the prediction at time $n+2$. Experimental results reported in Table 3 show that NN-RK2 actually achieves obviously more accurate prediction compared with Euler Method. By combining the multi-scale gridding method and NN-RK2, the simulation efficiency is greatly improved.

Method	Average pos error (mm)		Average distance to closest point d^n (mm)	Frame inference time (ms)
	n+1	n+2		
Results on DPI DamBreak				
DPI-Nets (Li et al.)	12.73	25.38	22.07	202.56
SPNets Convs (Schenck and Fox)	-	-	-	1058.46
PCNN Convs (Wang et al.)	0.72	1.67	19.79	187.34
SplineCNN Convs (Fey et al.)	0.71	1.65	170.20	67.67
KPConv (Thomas et al.)	2.49	7.05	unstable	47.96
CConv (Ummenhofer et al.)	**0.62**	1.49	**16.98**	12.55
CConv*	0.63	**1.46**	17.03	12.71
Ours (w/o NN-RK2)	0.74	1.78	18.79	6.78
Ours	0.74	1.91	20.05	**3.29**
Results on DFSPH data (6K particles)				
DPI-Nets (Li et al.)	26.19	51.77	unstable	305.55
SPNets Convs (Schenck and Fox)	-	-	-	784.35
PCNN Convs (Wang et al.)	0.67	1.87	32.51	319.17
SplineCNN Convs (Fey et al.)	0.68	1.93	unstable	281.92
KPConv (Thomas et al.)	1.65	4.54	unstable	57.89
CConv (Ummenhofer et al.)	**0.56**	1.51	**29.50**	16.47
CConv*	0.56	**1.50**	29.77	16.86
Ours (w/o NN-RK2)	0.68	1.96	31.04	8.39
Ours	0.68	2.12	32.98	**4.24**

Table 1: Results on DPI DamBreak and DFSPH dataset. One step, two steps and rollout prediction error are shown. Frame inference time of our method and others are shown in the table to prove that our method achieves significantly faster simulation. w/o NN-RK2 represents inference without using proposed NN-RK2 time integration. * denotes our reproduced results. All runtimes of our results are measured on a system with an Intel Xeon 6150 CPU and an NVIDIA RTX 2080Ti.

Experiments

We conduct experiments on multiple datasets and compare accuracy and inference time with prior works. Quantitative and qualitative results show that our method has a comparable fluid simulation capability with obviously faster inference speed. As the visualization results show, our method achieves high visual fidelity simulations. Moreover, we perform sufficient ablation studies to show the effectiveness of each component and setting of hyper-parameters.

Experiment Settings

Datasets DPI DamBreak data is generated with FleX, which is a position-based simulator that targets real-time applications.This data is used in both (Li et al.) and (Ummenhofer et al.). The scene simulates the behavior of a randomly placed fluid block in a static box. 2000 scenes and 300 scenes are generated for training and testing respectively. Experiments on a more challenging dataset used in (Ummenhofer et al.) are also conducted in our work. This dataset is generated with DFSPH (Bender and Koschier), which prioritizes simulation fidelity over runtime. DFSPH can generate accurate fluid flows with very low volume compression. The ground-truth data is generated by randomly placing multiple bodies of fluid in 10 different box-like scenes, which are shown in the appendix of (Ummenhofer et al.). 200 scenes and 20 scenes are generated for training and testing respectively. Following the setting in (Ummenhofer et al.), a simplified version with a constant number of particles (6,000) and a single box environment is generated for quantitative comparisons.

Evaluation Metrics The average error of the particle positions with respect to the ground truth is used to evaluate our method and others. Following the setting in (Ummenhofer et al.), we compute the deviation from the ground truth for two subsequent frames, denoted by $n+1$ and $n+2$. Besides, we also report the rollout error, *i.e.*, the average distance from the ground-truth particles to the closest particle in the prediction for the whole sequence, to measure long-term similarity.

Implementation Details We set the threshold δ used for generating multi-scale micelles for training data to 0.5. And the threshold ε for test data is set to 0.8. The minimum and the maximum number of particles of each micelle are set to 16 and 100 respectively. To limit the size of the micelles, we set the maximum depth of the Octree to 10 when generating multi-scale micelles for all data. We train our network for 50000 iterations with a batch size of 16 and an initial learning rate of 0.001. We half the learning rate at steps [20000, 25000, ..., 45000]. All these settings are the same in all our experiments unless otherwise noted.

Comparisons with Prior Works

We report our quantitative results involving prediction error and inference time in Table 1, where results of Continuous Convolution (Ummenhofer et al.) and other state-of-the-art methods are also shown. We observe that our method achieves comparable results to other state-of-the-art methods with significantly faster inference speed. For example,

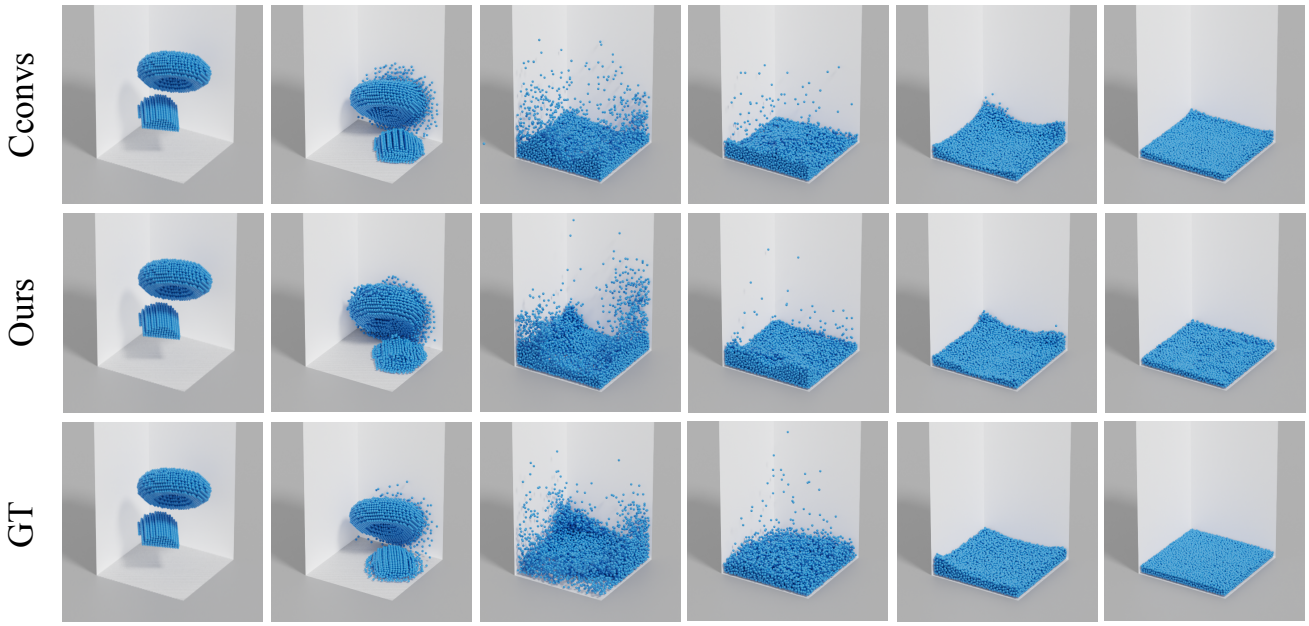

Figure 3: Some visualization results of simulation. Ground-truth and results of ours and CConv are all shown in the figure. Our method presents high visual fidelity simulations. See the videos in the supplementary for more simulation results.

our method shows a similar prediction error compared with PCNN Convs (Wang et al.), while whose inference time is about **75** times as ours. To prove that our method presents high visual fidelity simulation, we show some qualitative results of our method and CConvs in Figure 3.

Effectiveness of Dynamic Multi-Scale Gridding

To prove the effectiveness of our proposed dynamic multi-scale gridding algorithm, we present some visualization results of generated micelles in Figure 4 and conduct an ablation study to quantitatively prove it. For simplicity and intuition, we directly display the gravity centers of micelles to show the generated micelles in the Figure 4. By comparing the Lagrangian fluid samples with corresponding gravity centers of micelles, we see that our method generates more small scale micelles in areas where the relative motion of particles is intense (*e.g.*, the region where two objects collide) while generates a small number of large scale micelles in the areas where the particles move relatively uniformly. This indicates that our method does partition particles into multi-scale areas that share similar motion patterns. Moreover, we generate multi-scale micelles only according to the density of particles just the same as the naive Octree algorithm. To quantitatively analyze the effectiveness of our method, we present the results of the network trained with different micelles generation methods on the original DFSPH dataset. We set the maximal depth and the maximal number of particles to 10 and 100 respectively when generating micelles using the naive Octree algorithm. Results shown in Table 2 prove that our method achieves obviously better fluid simulation results.

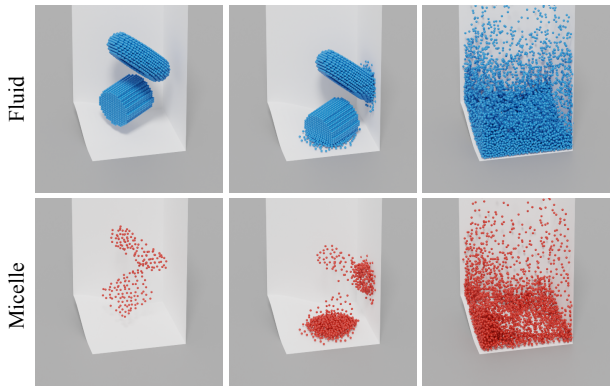

Figure 4: Visualization of our generated multi-scale micelles. The gravity centers of micelles are shown, whose density indicates the degree of motion variation.

Method	Average pos error (mm)	
	n+1	n+2
CConv (Ummenhofer et al.)	**0.67**	**1.87**
CConv*	**0.67**	1.88
Ours-Octree	1.02	2.93
Ours-Micelles	0.79	2.20

Table 2: Results on original version of DFSPH dataset. Ours-Octree denotes that we generate micelles only according to the density of particles.

Effectiveness of NN-RK2

By comparing results of ours and ours (without NN-RK2) in Table 1, we can conclude that our proposed NN-RK2 time

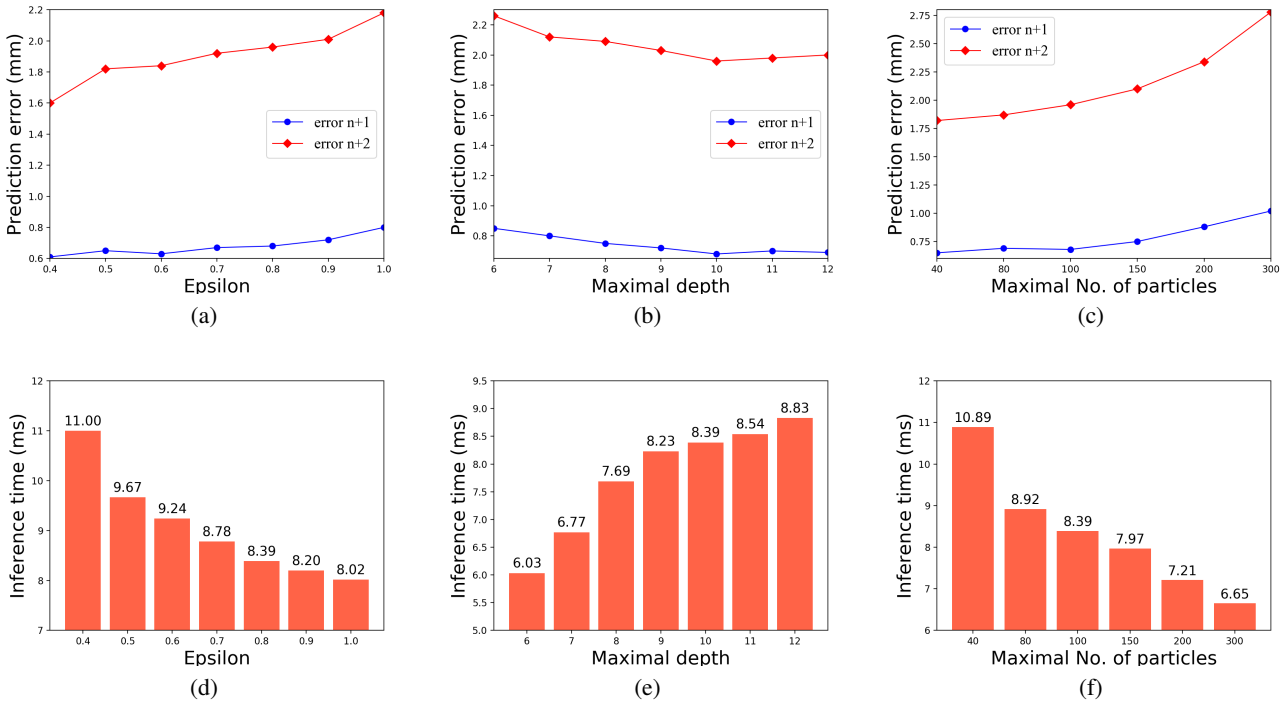

Figure 5: Results of parameter analyses performed on the simplified version of DFSPH data. Analysis of ε, the maximal depth and the maximal number of particles of micelle are shown in the subfigure a, b and c respectively, where one and two steps error are both shown. The corresponding results of inference time are shown in the subfigure d, e and f respectively.

Method	Average pos error (mm)	
	n+1	n+2
CConv (Euler with $2\Delta t$)	1.44	3.75
CConv (NN-RK2)	**0.56**	**1.79**

Table 3: Comparison between Euler and NN-RK2.

integration algorithm does accelerate inference and achieves approximately twice the speed on both two datasets. Moreover, one can make a trade-off between simulation accuracy and inference time by adjusting the order of Runge-Kutta algorithm. To demonstrate that our proposed time integration method does perform better than Euler method with doubled time-step, *i.e.*, train the network with doubled time-step and predict position and velocity of particles in the farther future. We report the results on Table 3, from where we observe that our method achieves significantly better performance. We think the more difficult task (*i.e.*, predict with doubled time-step) makes it hard for this light-weight model to predict accurately. And predicting larger numbers (larger displacement) makes the network unstable.

Parameter Analysis

There are some hyper-parameters introduced by our proposed multi-scale gridding algorithm. We analyze three of the most important parameters for our method, *i.e.*, ε used as a variance threshold for generating micelles during inference, the maximal depth that we set when generating micelles, and the maximal number of particles of each micelle we set when generating micelles. Experiments are conducted on the simplified version of DFSPH data, and the results are shown in Figure 5, where prediction error and the corresponding inference time are both shown. We observe that our method is robust to ε and the maximal depth, and we set ε to 0.8 and maximal depth to 10 by making a trade-off between accuracy and inference time. The experimental results also show that our method is relatively sensitive to the maximal number of particles of each micelle, *i.e.*, larger settings result in larger prediction error and too small settings result in too much inference time. In our work, we set it to 100 for DFSPH and 64 for DPI Dam Break.

Conclusion

We have developed a dynamic multi-scale gridding algorithm for fast fluid simulation. A Lagrangian fluid sample with an excessive amount of particles is divided into multi-scale micelles according to internal motion variation, thus reducing the magnitude of elements that have to be processed. By further modifying the second-order Runge-Kutta algorithm and combine with our method, we achieve high visual fidelity fluid simulations with an extremely fast speed.

Acknowledgements

This work was supported by National Science Foundation of China (U20B2072, 61976137). This work was also partially supported by Grant YG2021ZD18 from Shanghai Jiaotong University Medical Engineering Cross Research.

References

Battaglia, P. W.; Pascanu, R.; Lai, M.; Rezende, D. J.; and Kavukcuoglu, K. 2016. Interaction Networks for Learning about Objects, Relations and Physics. In *Advances in Neural Information Processing Systems 29: Advances in Neural Information Processing Systems 29, December 5-10, 2016, Barcelona, Spain*, 4502–4510.

Bender, J.; and Koschier, D. 2015. Divergence-Free Smoothed Particle Hydrodynamics. In *Proceedings of the 2015 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM.

Fey, M.; Lenssen, J. E.; Weichert, F.; and Müller, H. 2018. SplineCNN: Fast Geometric Deep Learning With Continuous B-Spline Kernels. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 869–877.

Holl, P.; Thuerey, N.; and Koltun, V. 2020. Learning to Control PDEs with Differentiable Physics. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Hu, Y.; Anderson, L.; Li, T.; Sun, Q.; Carr, N.; Ragan-Kelley, J.; and Durand, F. 2020. DiffTaichi: Differentiable Programming for Physical Simulation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Kim, B.; Azevedo, V. C.; Thuerey, N.; Kim, T.; Gross, M. H.; and Solenthaler, B. 2019. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Comput. Graph. Forum*.

Ladicky, L.; Jeong, S.; Solenthaler, B.; Pollefeys, M.; and Gross, M. H. 2015. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.*, 34(6): 199:1–199:9.

Li, Y.; Wu, J.; Tedrake, R.; Tenenbaum, J. B.; and Torralba, A. 2019. Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Monaghan, J. J. 1992. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30(1): 543–574.

Morton, J.; Jameson, A.; Kochenderfer, M. J.; and Witherden, F. D. 2018. Deep Dynamical Modeling and Control of Unsteady Fluid Flows. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 9278–9288.

Mrowca, D.; Zhuang, C.; Wang, E.; Haber, N.; Fei-Fei, L.; Tenenbaum, J.; and Yamins, D. L. 2018. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 8813–8824.

Müller, M.; Heidelberger, B.; Hennix, M.; and Ratcliff, J. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2): 109–118.

Pfaff, T.; Fortunato, M.; Sanchez-Gonzalez, A.; and Battaglia, P. W. 2021. Learning Mesh-Based Simulation with Graph Networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

Sanchez-Gonzalez, A.; Godwin, J.; Pfaff, T.; Ying, R.; Leskovec, J.; and Battaglia, P. W. 2020. Learning to Simulate Complex Physics with Graph Networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119, 8459–8468.

Schenck, C.; and Fox, D. 2018. SPNets: Differentiable Fluid Dynamics for Deep Neural Networks. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*.

Solenthaler, B.; and Pajarola, R. 2009. Predictive-corrective incompressible SPH. *ACM Trans. Graph.*, 28(3): 40.

Sulsky, D.; Zhou, S.-J.; and Schreyer, H. L. 1995. Application of a particle-in-cell method to solid mechanics. *Computer physics communications*, 87(1-2): 236–252.

Thomas, H.; Qi, C. R.; Deschaud, J.; Marcotegui, B.; Goulette, F.; and Guibas, L. J. 2019. KPConv: Flexible and Deformable Convolution for Point Clouds. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 6410–6419.

Tompson, J.; Schlachter, K.; Sprechmann, P.; and Perlin, K. 2017. Accelerating Eulerian Fluid Simulation With Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70, 3424–3433.

Ummenhofer, B.; Prantl, L.; Thuerey, N.; and Koltun, V. 2020. Lagrangian Fluid Simulation with Continuous Convolutions. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Wang, S.; Suo, S.; Ma, W.; Pokrovsky, A.; and Urtasun, R. 2018. Deep Parametric Continuous Convolutional Neural Networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2589–2597.