Graphical Models 000 (2020) 101061

Contents lists available at ScienceDirect

Graphical Models

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

Dynamic particle partitioning SPH model for high-speed fluids simulation

Yang Gao [a,b,1], Zhong Zheng [b,1], Jin Li [b], Shuai Li [b,c,*], Aimin Hao [a,b,c], Hong Qin [d]

[a] Research Institute for Frontier Science, Beihang University, China
[b] State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, China
[c] Peng Cheng Laboratory, Shenzhen, China
[d] Department of Computer Science, Stony Brook University, USA

ARTICLE INFO

Keywords:
Computing methodologies
Animation
Physical simulation

ABSTRACT

The popular SPH method still remains as one of the most widely-used methods in fluid simulation, exhibiting its longevity with new and diverse variants in recent decades. New progress in the SPH simulation in most recent years are still hampered by such challenge when simulating high-speed fluids. In this paper, our research efforts are devoted to the efficiency issue of the SPH simulation when the ratio of velocities among fluid particles is large. Specifically, we introduce a *k*-means clustering method into the SPH framework to dynamically partition fluid particles into two disjoint groups based on their velocities. Then, we use a two-scale time-step scheme for these two types of particles. The smaller time steps are for particles with higher speed in order to preserve temporal details and guarantee stability. In contrast, the larger time steps are used for particles with smaller speed to reduce the computational expense, and both types of particles are tightly coupled in the simulation. We conduct various experiments and compare our method with some of the most relevant works, which have manifested the advantages of our methods over the conventional SPH technique and its new variants in terms of efficiency and stability.

1. Introduction and motivation

Over the past several decades, the smoothed particle hydrodynamics (SPH) method has proved to have a strong ability of to simulate a wide variety of fantastic fluid phenomena. In SPH methods, fluids are essentially discretized as a number of particles, which carry some physical properties such as mass, velocity, and position augmented and blended by certain kernel functions, so SPH methods could naturally guarantee the conservation of mass. Augmented by the implicit tracking of frequent topology changes without having an explicit grid structure or meshes, SPH methods have demonstrated an advantage over other fluid simulation methods. The standard SPH method [24] is suitable for the simulation of compressible, but will meet a challenge when simulating incompressible fluids. In recent years, many variants of SPH methods have been developed to enable the ability for simulating some fantastic phenomena. WCSPH [3] introduces the Tait equation to SPH and enforces weakly compressibility on fluids, but it is realized at the cost of limiting time step size, thus becomes inefficiency on overall simulation. PCISPH [32] proposes a prediction-correction scheme to resolve pressure forces and enforces incompressibility and significantly improves the performance. IISPH [17] obtains a discretized form of the pressure Poisson equation and improves the convergence rate of the pressure solver, thus allows a larger time step and small density deviations. DF-SPH [4] proposes two implicit pressure solvers to enforce a low-density compression and a divergence-free velocity field. Those implicit solvers have greatly improved the efficiency of SPH simulations, but pressure solving remains one of the most time-consuming parts during the simulation.

For highly turbulent fluid simulations, especially those containing particles with large velocity differences, the huge velocity differences would easily give rise to unstable simulation or result in some artifacts. Considering an example of a high-pressure water gun spraying water into the pool. Furthermore, the high-speed fluids usually generate lots of splashes and sprays, having much more details than the rest of near-normal fluids in reality. For most existing SPH models, high-speed fluid animation may behave abnormally due to the numerical instability, or miss the interesting details generated by severe collision. Using smaller time steps may relieve the instability issues and keep the splash details, which is at the expense of time efficiency. It may be noted, the inefficiency will become especially obvious as the number of particles increases. To tackle these issues, we aim to propose a robust and efficient model for stable high-speed fluids simulation with sufficient and vivid details.

Most of the current methods are either using a unified constant time, or adopting an adaptive one which is directly derived from the CFL condition by using the maximum velocity of all particles. They all apply

* Corresponding author at: Haidian Street, Xueyuan Road, No. 37, Beijiang, China.
E-mail address: lishuaiouc@126.com (S. Li).
[1] Contributed equally and should be regarded as co-first authors.

https://doi.org/10.1016/j.gmod.2020.101061
Received 17 August 2019; Received in revised form 23 January 2020; Accepted 19 February 2020
Available online xxx
1524-0703/© 2020 Published by Elsevier Inc.

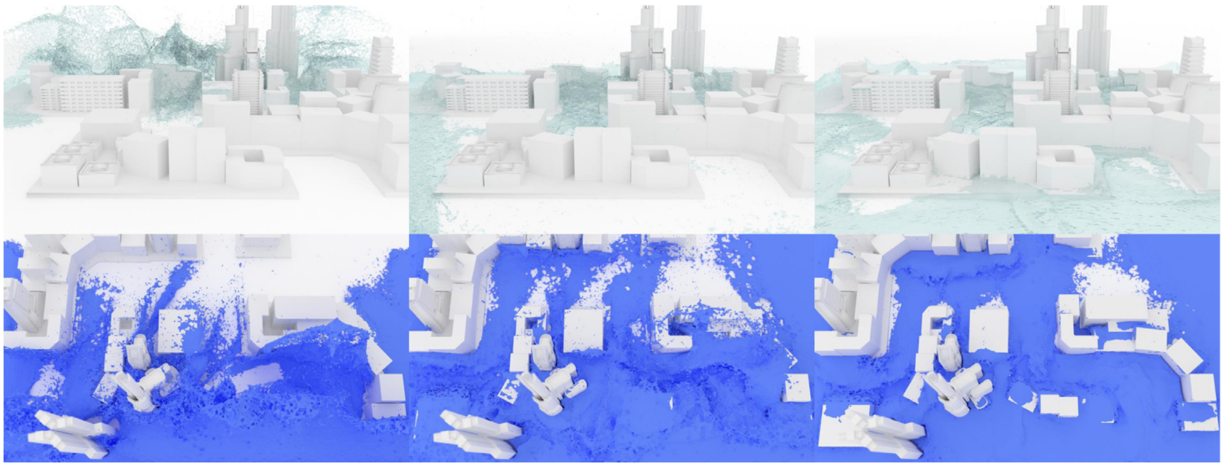
Fig. 1. City blocks with flooding.

the constant time step to all the particles at the same moment. To address the efficiency issues in turbulent fluid simulation, we propose a dynamic particle partitioning SPH method (PPSPH) that improves the simulation's efficiency through changing the time step size for different fluid particles. We divide each simulation time step into certain numbers of small sub-steps for particles with high velocities, and in each sub-step, we simplify the computational procedure when computing the particles that have low velocities, and then promise a correct final result at the end of each major step. Specifically, the salient contributions of this paper include:

- We introduce a *k*-means clustering scheme into SPH particles partition based on their velocities as a pre-treatment measure for our dynamic time step model.
- We propose a dynamic two-scale time step model for particles with a large difference in velocities. A small-time step for fast particles ensures stability and sufficient details, while large time step for slow ones improves the efficiency.
- Our method is easy to be performed on most of the existing SPH frameworks and can be implemented on GPU for high-efficiency applications.

In practice, our method could function well within any popular SPH framework, and in this paper, we integrate our method with PCISPH [32] and DFSPH [4] for the comparison purpose, and our visually-detailed fluid simulations are based on DFSPH.

2. Related works

In recent years, a lot of visual-effect-dominant fluid simulation methods have been introduced for visually-appealing animation. For example, fluid implicit particle (FLIP) method [44] is a popular particle-grid coupling method to simulate multiphase fluid [5,10], splashing water [11,43], fluid-solid coupling [9], etc. Lattice Boltzmann method (LBM) [13,38] uses Eulerian grid solving fluid animation in a physically-based way. Also, data-driven methods [19,29] and position based fluid [20,21] all achieve good achievements. However, among the fluid simulation methods, SPH may be the most widely used and most studied method.

The fluid simulation method of Müller et al. [25] was based on the equation of state to weakly enforce incompressibility. It was well suited for some low-speed compressible fluid simulations. For example, Adams et al[1]. computed pressure in an efficient and straightforward way. Later, WCSPH [3] was proposed to enforce a weakly compressible form of SPH for fluid flow based on the Tait equation. For low-speed fluid simulation, viscosity is one of the important attributes of fluids, and high viscosity fluids like honey and mud have different characteristics from water. XSPH was introduced in [23,24] and adopted in [31] to simulate fluids with low viscosity. Also, there are some other approximate methods [33] computing a viscosity force from relative velocities to smooth the velocity field. He et al. [15] proposed a local Poisson SPH Method to solve viscous incompressible fluids and preserve the advantages of WCSPH and ISPH. For non-Newtonian fluids, viscosity can be modeled by the deformation tensor [26]. Takahashi et al. [34] used an implicit formulation of viscosity adapted from Eulerian formulation [2] to improve the robustness and efficiency of viscosity integration. Peer et al. [27,28] proposed a method based on prescribing a target gradient and reconstructing the velocity field to simulate high viscosity fluids. And then Weiler et al. [39] provided several quantitative and qualitative comparisons of the SPH-based implicit viscosity methods and presented a novel method for the simulation of highly viscous fluids which surpasses the existing approaches in physical accuracy and visual realism.

Unlike viscous SPH fluids, turbulent fluid simulation always focuses on pressure solving. Implicit pressure solvers were introduced by Solenthaler et al. in PCISPH [32], which computed the pressure field by solving a linear system to approximate the rest density and thus enforced a low-density variation. Then Macklin et al. [20] also used an iterative correction method to correct particle positions to enforce incompressibility. Projection schemes can also be implemented in SPH, and pressure Poisson equation is solved to compute pressure, though it is usually used in some Euler approaches [6,8]. Ihmsen et al. proposed IISPH [17], a formulation of the projection method for SPH allowing a larger time step and exhibiting a superior convergence rate than PCISPH, which could relatively simulate high-speed fluid animation. And Bender et al. [4] employed two solvers to enforce a constant density condition and a divergence-free, and improved the stability and performance while simulating turbulent fluids. Ihmsen et al. [18] summarized the state-of-the-art researches within the graphics community. Some asynchronous methods [16,30,35] have been proposed to boost the efficiency of SPH simulations, of which Reinhardt et al. [30] proposed an approach of dedicated time steps for each particle and introduced parallelization queues for asynchronous time integration. It could accelerate up to a factor of 7.5 compared with non-iterative EOS solvers with the fixed time stepping method. However, this solver is impossible to use iterative SPH models because of the lack of global convergence criteria, while our novel method can be conveniently applied to the foremost methods such as PCISPH, IISPH, and so on to improve the efficiency and performance greatly.

At the same time, machine learning has recently received widespread attention in the field of fluid simulation. The combination of fluid solutions and machine learning techniques was first proposed by [19]. Chu et al. [7] and Xie et al. [40] used a convolutional neural network to generate super-resolution results from low-resolution simulation scenarios. Um et al. [37] proposed a deep neural network to add a large number of splashed droplet details to a low-resolution simulation scene. Moreover, Yang et al. [41] and Tompson et al. [36] accelerated the solving of the Poisson equation by training a neural network as the pressure solver. Eckert et al. [22] proposed large-scale data set of reconstructions of smoke plumes, which enabled the reconstructed velocities for further applications such as re-simulation, flow enhancement, etc.

3. Method overview

In this paper, we propose a particle partition SPH method (PPSPH) to simulate particles with different speeds by dynamic time steps. Compare to the traditional SPH framework, our simulation step consists of three main components: (1) Partitioning particles into two disjoint groups by k-means clustering; (2) Uncoupling high-speed particles from fluids and simulating faster particles with refined time-step size; And (3) Integrating all fluid particles including faster ones and slower ones.

3.1. SPH fundamentals

In the concept of SPH, the quantity A_i at location $\mathbf{x}_i$ is interpolated by a weighted sum of A_j from neighboring particles

$$A_i = \sum_j A_j \frac{m_j}{\rho_j} W(\mathbf{x}_i - \mathbf{x}_j, h),$$

where m_j and ρ_j are the mass and density of particle j separately, W is the smoothing kernel function with support of radius h. The density ρ_i is approximated as

$$\rho_i = \sum_j \rho_j \frac{m_j}{\rho_j} W(\mathbf{x}_i - \mathbf{x}_j, h) = \sum_j m_j W(\mathbf{x}_i - \mathbf{x}_j, h).$$

The derivative of quantity A_i only affects the smoothing kernel

$$\nabla A_i = \sum_j A_j \frac{m_j}{\rho_j} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h),$$

and the pressure field is derived from ideal gas state equation

$$p_i = \kappa(\rho_i - \rho_0),$$

where ρ_0 is the rest density of fluids. In incompressible fluid simulations, the improved models such as PCISPH, IISPH, or DFSPH usually use implicit pressure solvers to guarantee density deviation at a low level, which can also improve the efficiency of the simulation.

3.2. Algorithmic outline

Distinct from the existing SPH methods using the unified (constant or adaptive) time step for all particles, we focus on the refined processing of particles at different speeds. According to the CFL condition, the maximum simulation time step is limited to the inverse of the maximum velocity of all particles, so the time step should be much smaller if the scenarios consist of high-speed particles. It will inevitably lead to inefficiency when particles have large differences in velocities and have an unbalanced ratio on the number of different types of particles at the same time. To cope with the efficiency issue and keep vivid details of high-speed fluids, we propose a new simulation scheme for SPH methods, which uncouples high-speed particles from the whole fluids by k-means clustering. High-speed particles have a faster motion than the rest of fluid particles, thus the time step derived from the CFL condition is adapted to ensure the stability. To reduce computation we use different time-step sizes and give high-speed particles more chances for detail capture, and then integrate all particles after processing certain times of simulation steps.

After partitioning high-speed particles dynamically, we divide a complete simulation time step into various major steps, and each major step consists of a certain amount of mini-steps. In the first few mini-steps of the major step, we only calculate high-speed particles and use the maximum velocity of all particles to derive time-step size from the CFL condition to ensure numerical stability. Low-speed particles will rest until we arrive at the last mini step. We integrate the high-speed particles with low-speed particles to simulate the motion for all particles in the scene. In the integration step, we make up the time step for low-speed particles to cope with the lack of motion of those low-speed particles in previous mini steps. Algorithm 1 outlines the main scheme of our method, where N_{mini} is the number of mini steps in a major step and *iter* is the current mini-steps, λ is a user-defined value to determine whether k-means clustering procedure should be proceeded, and $v^*_{\max}$ is the maximum velocity of all particles. Fig. 2 also illustrates the main procedure of our method.

Algorithm 1 Simulation procedure.

```
1:  set the amount of mini steps in major step $N_{mini} = 0$
2:  set the index of mini step in major step $iter = 0$
3:  while simulation do
4:      for all particle $i$ do
5:          find neighborhoods
6:      end for
7:      for all particle $i$ do
8:          compute densities $\rho_i$
9:          compute non-pressure forces $\mathbf{F}_i^{ext}$
10:         predict velocities $\mathbf{v}_i^*$
11:     end for
12:     if $iter \geq N_{mini}$ then
13:         if $v^*_{\max} < \lambda$ then
14:             derive time step size $\Delta t$ from CFL condition
15:             update $N_{mini} = 1$
16:         else
17:             partition particles based on $\mathbf{v}_i^*$
18:             compute mini and major time step sizes
19:             update $N_{mini}$
20:         end if
21:     else
22:         if $iter < N_{mini} - 1$ then
23:             solve pressure for high-speed particles
24:             update velocity and position for high-speed particles
25:         else
26:             solve pressure for all particles
27:             update velocities and position for all particles
28:         end if
29:     end if
30: end while
```

4. Particle classification

We implement different strategies on fluid particles belong to different groups to improve the efficiency of the simulation. In this section we will introduce our particle partitioning procedure. In our method, all fluid particles will be partitioned into two different clusters based on their velocities. To fulfill the requirement, we introduce the k-means clustering algorithm [14] to our method as the particle partitioning method. Applying a threshold manually might not be a suitable way compared with our automatic k-means classification. A fixed threshold can not adjust automatically with the velocities change in a scenario, which might cause unreasonable classification. What is more, a fixed threshold may not be appropriate in every scenario, for different simulations it may need to be reset manually for different ranges of velocities. Therefore, our work addresses these points by introducing the k-means clustering algorithm into our particle partitioning procedure. In our method, all fluid particles will be partitioned into two different clusters dynamically based on their velocities in every major time step. Regardless of the difference in velocity distribution for different scenarios, our method always finds the proper classification automatically,

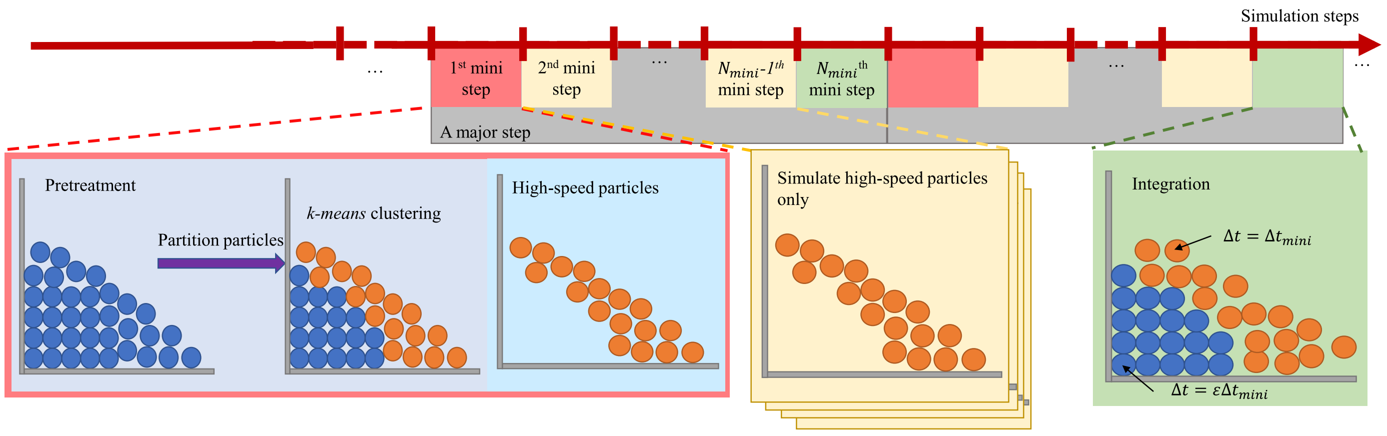

Fig. 2. A brief illustration of our method.

which is quite suitable to be implemented for its fast convergence and easy deployment.

4.1. k-*means clustering*

k-means clustering is quite suitable to be implemented in our method for its quick convergence and easy deployment. As a popular method in data mining for data cluster analysis, and its function is partitioning *n* points into *k* clusters. The goal of the algorithm is to minimize the following within-cluster sum of squares

$$\arg\min \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} (\mathbf{x}_j - \bar{\mathbf{x}}_{S_i}),$$

where S_i is the i_{th} cluster containing points which are closer to the center value of the i_{th} cluster than any other clusters and $\bar{\mathbf{x}}_{S_i}$ is the mean value of all points in the i_{th} cluster calculated in the previous iteration. For the first iteration in *k*-means clustering, it can be a randomly generated value or a user-defined value.

4.2. *Data clustering*

We use particle velocities as sample observations to partition particles into two different clusters. After processing the clustering, we use the maximum value of each cluster to derive the time-step size from the CFL condition, and compute the ratio between the major and mini-steps size to determine the number of mini-steps (N_{mini}), and combine these mini steps to be a major step. According to the CFL condition, we need to compute scalar lengths of particle velocity vectors as sample observations in *k*-means clustering.

As the observation of *k*-means clustering, the predicted velocity is derived from the final velocity in the previous iteration and the non-pressure forces

$$\mathbf{v}_i^* = \mathbf{v}_i + \Delta t \frac{\mathbf{F}_i^{ext}}{m_i}. \qquad (1)$$

The *k*-means clustering usually converges to a local optimal value, and the result is prone to fluctuation (be affected by the initial mean value of each cluster). In order to have a more clear dividing line for the particles, our method usually sets the maximum predicted velocity to the initial mean value of one cluster, and sets the threshold value to another cluster (i.e., symbol λ in Algorithm 1). The threshold value is a user-defined value used to determine whether the *k*-means clustering procedure will be proceeded. we set a threshold pretreatment because at the beginning of the simulation or in some other special scenes, all the particles velocities are at a low level. The time-step size derived from the CFL condition is large enough and there won't be a significant improvement on the efficiency by using our clustering method, thus the clustering procedure will not be proceeded in such circumstance (which means our PPSPH method can recognize fluid scenes and only works on high-speed fluid scenes). In the end of the clustering, particles will be partitioned into two clusters, one containing particles with slower velocities, another with faster velocities.

Please note, the clustering and the time-step updating procedures are only performed in the first mini step of every major time step. Newly generated particles, if any, will be simply partitioned into the high-speed particle cluster if their velocity is larger than the maximum velocity of the low-speed cluster and vice versa.

A complete clustering procedure is illustrated as Algorithm 2.

Algorithm 2 Implementation of *k*-means clustering for particle partition based on particle velocities.

```
1:  for all particle $i$ do
2:      predict particle velocity $v_i^* = \left\| \mathbf{v}_i + \Delta t \mathbf{F}_i^{ext} / m_i \right\|_2$
3:  end for
4:  calculate maximum velocity of all particles $v_{\max}$
5:  if $v_{\max} > \lambda$ then
6:      initialize mean value of two clusters: $\bar{x}_0 = v_{thres}$, $\bar{x}_1 = v_{\max}$
7:      compute distance and update cluster index for all particles
8:      while clustering not converged do
9:          compute mean value of velocity for each cluster
10:         update $\bar{x}_0$, $\bar{x}_1$
11:         update cluster index for all particles
12:         compute differences on cluster mean values with previous values
13:     end while
14:     calculate the maximum velocity for each cluster
15:     store particle clustering results
16: else
17:     store the maximum predicted velocity of all particles
18: end if
```

5. Simulation

Section 4 discusses the implementation of *k*-means clustering in our method to partition particles into two groups based on particle velocities. In our method, a major step consists of N_{mini} times of mini steps. At the beginning of each major step, a particle partitioning procedure will be processed and the clustering result will be stored. We store the maximum velocity of each cluster. The cluster labels particles which it currently contains, and the particle index set of the high-speed cluster is also stored.

5.1. Major step

In each mini step of the major step except the last one, we only solve pressure and update velocities and positions for particles with a higher velocity. That is, in this period, slow particles don't participate in the calculations and are *waiting* for the high-speed particles. But in the last mini step, we will implement a complete simulation procedure for all particles in the scene like the standard SPH methods, but with some modifications.

So when simulating incompressible fluids with our method, we only guarantee the incompressibility for high-speed particles at the end of a mini time step. And we guarantee that the incompressible condition is completely fulfilled for all particles at the end of every major time step, as we process pressure solving for all fluid particles in the last mini step.

A major step consists of N_{mini} mini-steps, but the major time-step size is not a simple multiple of mini time-step size in most cases during the simulation since all the time steps are dynamically adaptive. Particles with high velocity will be processed in each mini time step, and we should guarantee that each particle has a normal motion to avoid artifacts like penetration in mini-steps, so we choose the maximum velocity of all fluid particles to derive the mini time step, and the maximum velocity of particles in the low-speed group to derive the major time step from the CFL condition. According to the CFL condition

$$\Delta t \leq 0.4 \frac{d}{\|\mathbf{v}_{max}\|_2}, \tag{2}$$

where d is the diameter of the particle, applying the maximum velocity of all particles $\mathbf{v}_{max,all}$ and the maximum velocity of particles in low-speed group $\mathbf{v}_{max,\ low}$ to Eq. (2). We obtain the major and the mini time step size

$$\Delta t_{mini} \leq 0.4 \frac{d}{\|\mathbf{v}_{max,all}\|_2}, \tag{3}$$

$$\Delta t_{major} \leq 0.4 \frac{d}{\|\mathbf{v}_{max,low}\|_2}. \tag{4}$$

Also, we set a user-defined threshold (N_{tres}). When the ratio is smaller than the threshold, or more intuitively speaking, the particle velocities are close to each other and don't have a large difference, the major step will only contain one standard SPH time step. To prevent the ratio from being too large which may cause an unreal simulation result, we set a maximum mini time step amount $N_{\max}$. The amount of mini step in a major step is computed as

$$N_{mini} = \begin{cases} \min(N_{max}, \left\lceil \frac{\Delta t_{major}}{\Delta t_{mini}} \right\rceil) & \text{if } \frac{\Delta t_{major}}{\Delta t_{mini}} \geq N_{tres} \\ 1 & \text{else} \end{cases}, \tag{5}$$

where we set the threshold N_{tres} in Eq. (5) to a number less than 1.5. After determining the amount of mini steps N_{mini}, different strategies will apply on the following mini steps.

5.2. Mini step

We have mentioned in the previous section that the mini step is a part of the major step.In our method, during the first $N_{mini} - 1$ mini steps, only particles partitioned into the high-speed group will process pressure solving and update their velocities and positions, because we have noticed that the pressure solving procedure is the most time-consuming procedure during the whole simulation step. The velocities and positions of slow particles are not computed during the first $N_{mini} - 1$ mini steps of major steps. In the last mini step of a major step, all particles in the scene will process pressure solving and update their positions. That is to say, slow particles will *wait for* fast particles and couple with fast particles until $N_{mini}th$ mini step

Our method can be conveniently integrated with the existing SPH frameworks. For PCISPH, the procedure of the first $N_{mini} - 1$ mini-steps is summarized as Algorithm 3 shows. ρ_{err} is the average error of density controlled by parameter η. Procedure before the pressure solving is the same as the standard PCISPH algorithm. We perform neighborhood search, and compute densities for all particles in the scene, but we don't recompute the time step in every mini step.

Algorithm 3 Mini step for high-speed particles in PCISPH.

```
1: while ($\rho^*_{err} > \eta$) || ($iter$<minIterations) do
2:     for all particle $i$ in high-speed particle cluster do
3:         predict velocity $\mathbf{v}^*_\mathbf{i}$
4:         predict position $\mathbf{x}^*_\mathbf{i}$
5:     end for
6:     for all particle $i$ in high-speed particle cluster do
7:         predict density $\rho^*_i$ and density error $\rho^*_{err}$
8:         update pressure $p_i$ and pressure force $\mathbf{F}^p_i$
9:     end for
10: end while
11: for all particle $i$ in high-speed particle cluster do
12:     compute new velocity $\mathbf{v}'_i$
13:     compute new position $\mathbf{x}'_i$
14: end for
```

Similarly, we also implement our method on DFSPH framework, and summarize the procedure of the first $N_{mini} - 1$ mini steps in Algorithm 4, where η^{div} is a threshold controlling the average density change rate. Different from PCISPH, the standard DFSPH framework adapts time-step size after processing the divergence-free solving procedure. In our method, we do it before the divergence solving, thus we can also apply our scheme on divergence solving procedure to further improve efficiency.

Algorithm 4 Mini step for high-speed particles in DFSPH.

```
1: for all particles $i$ do
2:     find neighborhoods
3: end for
4: for all particles $i$ do
5:     compute density $\rho_i$
6:     compute factor $\alpha_i$
7: end for
8: while $\left(\left(\frac{D\rho}{Dt}\right)_{avg} > \eta^{div}\right)$||($iter < 1$) do
9:     for all particle $i$ in high-speed particle cluster do
10:         compute $\frac{D\rho_i}{Dt}$
11:     end for
12:     for all particle $i$ in high-speed particle cluster do
13:         adapt velocity $\mathbf{v_i}^*$
14:     end for
15: end while
16: for all particle $i$ in high-speed particle cluster do
17:     compute non-pressure force $\mathbf{F}^{ext}_i$
18: end for
19: while $(\rho_{avg} - \rho_0 > \eta^\rho)$||($iter < 2$) do
20:     for all particle $i$ in high-speed particle cluster do
21:         compute density $\rho^*_i$
22:     end for
23:     for all particle $i$ in high-speed particle cluster do
24:         adapt velocity $\mathbf{v_i}$
25:     end for
26: end while
27: for all particle $i$ in high-speed particle cluster do
28:     update position $\mathbf{x}'_i$
29: end for
```

Since our method can be easily integrated into the state-of-the-art SPH simulation pipelines, we take DFSPH as an example. During the first $N_{mini} - 1$ mini steps of every major step, the current divergence error in fast particle i_f is determined using the DFSPH formulation,

$$\frac{D\rho_{i_f}}{Dt} = -\rho_{i_f} \nabla \cdot \mathbf{v}^*_{i_f}, \tag{6}$$

where $D(\cdot)/Dt$ denotes the material derivative, ρ_{i_f} and $\mathbf{v}^*_{i_f}$ denote density and intermediate velocity of fast particle i_f respectively.

According to DFSPH, the intermediate velocity of fast particle $v^*_{i_f}(t + \Delta t_{mini})$ and density $\rho^*_{i_f}(t + \Delta t_{mini})$ at time $t + \Delta t_{mini}$ are computed using Eqs. (7)–(9).

$$\mathbf{v}^*_{i_f}(t + \Delta t_{mini}) = \mathbf{v}^*_{i_f}(t) - \Delta t_{mini}\left(\sum_j m_j \left(\frac{k^{\mathbf{v}}_{i_f}(t)}{\rho_{i_f}(t)} + \frac{k^{\mathbf{v}}_{j_f}(t)}{\rho_{j_f}(t)} + \frac{k^{\mathbf{v}}_{j_s}(t)}{\rho_{j_s}(t)}\right)\nabla(W_{i_f j_f} + W_{i_f j_s})\right), \tag{7}$$

$$k^{\mathbf{v}}_{i_f}(t) = \frac{1}{\Delta t_{mini}} \frac{D\rho_{i_f}(t)}{Dt} \alpha_{i_f}(t), \tag{8}$$

$$\rho^*_{i_f}(t + \Delta t_{mini}) = \rho_{i_f}(t) + \Delta t_{mini} \sum_j m_j (\mathbf{v}^*_{i_f}(t + \Delta t_{mini}) - \mathbf{v}^*_{j_f}(t + \Delta t_{mini}) - \mathbf{v}^*_{j_s}(t))\nabla(W_{i_f j_f} + W_{i_f j_s}), \tag{9}$$

here, j denotes the neighborhood particle no matter whether it is fast or slow particle, m_j denotes the mass of particle j, $k^{\mathbf{v}}_{i_f}(t)$ is the stiffness parameter of high-speed particle i_f at time t, and $\alpha_{i_f}(t)$ represents a factor that solely depends on the current particle position. It may be noted that, $W_{i_f j_s} = W(x_{i_f} x_{j_s}, h)$ is a smoothing kernel function with supporting radius h. What we wish to emphasize here is that, $k^{\mathbf{v}}_{j_s}(t)$, $v^*_{j_s}(t)$ and x_{j_s} are equal to the values calculated in the previous major step, because we don't recompute them in the first $N_{mini}1$ mini steps of every major step.

Since during the first $N_{mini} - 1$ steps, low-speed particles are *waiting* for the high-speed particles, to make up the difference in these two-scale time steps. We propose a solution of having an adjustment on the time step for low-speed particles:

$$\Delta t_i = \begin{cases} \epsilon \Delta t_{mini} & \text{if particle } i \in \text{low-speed cluster} \\ \Delta t_{mini} & \text{else} \end{cases}, \tag{10}$$

where ϵ is a user-defined parameter of time compensation of low-speed particles in the last mini step of a major step. The parameter ϵ is usually set to a number larger than 1 in our method. When it is set to 1, it will be similar to the circumstance that high-speed particles move in each mini time step, and low-speed particles also move for 1 mini step per major step. The higher the value ϵ is, the more temporal details in the simulation will be produced and it will spend more computational time. And ϵ should not be set to a number larger than a value the CFL condition indicates. In our implementation, we usually set the number to 1.5 to ensure the equilibrium of detailed description and the efficiency when $N_{\max} \leq 3$. The connection between value ϵ and time efficiency will be discussed in the following section.

In the last mini step of a major step, we process the complete SPH time step procedure for all particles in the scene, but with a few revisions. It is only when velocity and position are updated that slow particle i_s and fast particle i_f are calculated separately due to the time step difference, so we update the velocity and position of particle i_s at time $t + \epsilon\Delta t_{mini}$ and i_f at time $t + \Delta t_{mini}$ in Eq. (11) to Eq. (14) respectively,

$$v_{i_s}(t + \epsilon\Delta t_{mini}) = v_{i_s}(t) + a\epsilon\Delta t_{mini}, \tag{11}$$

$$v_{i_f}(t + \Delta t_{mini}) = v_{i_f}(t) + a\Delta t_{mini}, \tag{12}$$

$$x_{i_s}(t + \epsilon\Delta t_{mini}) = x_{i_s}(t) + \epsilon\Delta t_{mini} v_{i_s}(t + \epsilon\Delta t_{mini}), \tag{13}$$

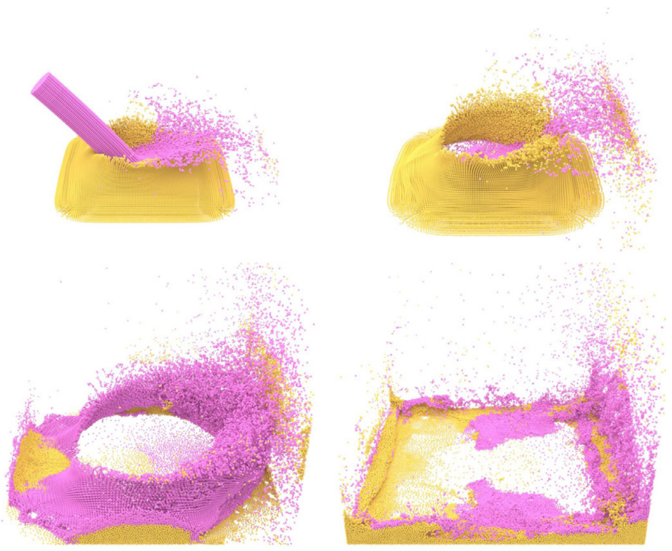
Fig. 3. Particle partition. From top to bottom: High-speed fluids interact with a fluid block. Particles partitioned into the low-speed group is colored in yellow, and the ones partitioned into the high-speed group is colored in pink. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$$x_{i_f}(t + \Delta t_{mini}) = x_{i_f}(t) + \Delta t_{mini} v_{i_f}(t + \Delta t_{mini}), \tag{14}$$

here, a denotes the acceleration of all particles, which is updated in this mini step. In addition, $v_{i_s}(t)$ denotes the velocity of low-speed particles which is equal to the value calculated in the previous major step, while $v_{i_f}(t)$ denotes the velocity of high-speed particles in the previous mini step.

6. Experimental results and discussion

In this paper, we integrate our method with both PCISPH and DFSPH frameworks, and employ the cubic spline kernel in simulation and use XSPH variant [31] to compute viscosity terms for low viscosity fluids. Performances are measured on an NVIDIA GTX 1070 graphic card with 8GB VRAM and an 8-core 3.40 GHz Intel i7 with 24 GB RAM. The images are rendered with Blender. All time measurements don't include the rendering procedure. We fully implement the simulation procedure of our methods on CUDA for efficiency. Also, please refer to our supplementary video for more vivid animations with details.

6.1. Particle partition

Fig. 3 shows a scenario of high-speed fluids flushing to a fluid block on the ground to demonstrate our dynamic particle partitioning procedure. The speed of pink fluids is much higher than the fluid block on the ground at the initial stage, so the partition result separates to both groups of fluids distinctly. After fluids interacting with each other, some high-speed fluids become slower and some low-speed fluids become faster. The cluster index which labels particles is dynamically adjusted as their velocities are changing with the simulation. Because of the features of k-means clustering, some particles with an intermediate-level speed are clustered into the high-speed group as the 3*rd* picture illustrates, and later some of them are clustered into the low-speed group since their velocities are close to the critical value.

Fig. 4 shows a verticle emitter under water, and Fig. 5 a horizontal emitter. We set ϵ to 2 to ensure the balance of detailed description and the efficiency. Both scenarios have exhibited good performance, and we can hardly notice artifacts.

Table 1
Performance of *k*-means clustering.

Number of particles	350k	550k	1.2 m	3.0 m
k-means clustering	29.1 ms	40.04 ms	64.4 ms	138.6 ms
Time step update (per mini step)	8.48 ms	10.5 ms	29.1 ms	79.0 ms

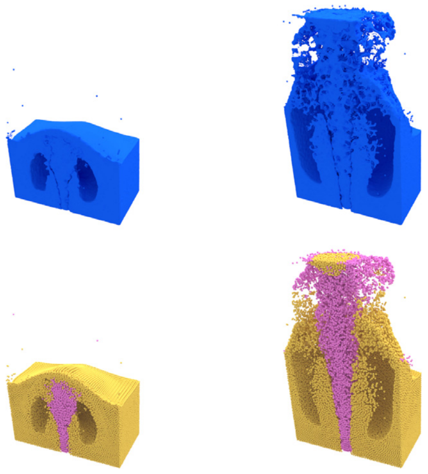

Fig. 4. The result of having a verticle emitter under water.

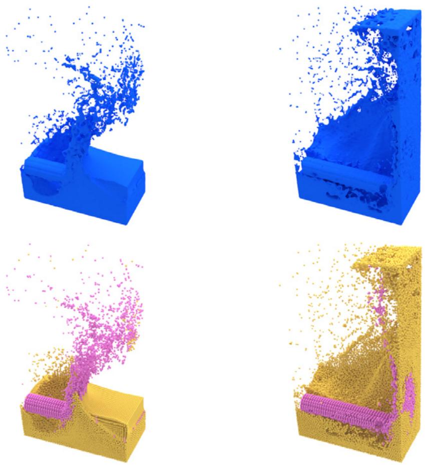

Fig. 5. The result of having a horizontal emitter under water.

Table 1 compares the efficiency of *k*-means clustering procedure in our method (PP-SPH) with the time step updating procedure of standard SPH methods. Note that, in this measurement, *k*-means clustering also includes the time step computing procedure. Although clustering costs some time, we process the clustering procedure only once per N_{mini} step. It has obvious advantages when simulating high-speed fluids, since the greater the speed difference is, the larger N_{mini} will be, and the better performance we could have than the standard ones.

6.2. Performance

We simulate the scenes with an adaptive time-step scheme, where the time step is derived from the CFL condition at the beginning of each major step. Unless specifically mentioned, in our experiments that contain *k*-means clustering procedure to partition particles, we set the maximum mini-steps number $N_{\max}$ in a major step to 3. And set the parameter ϵ used to adjust time-steps for low-speed particles to 1.5.

Table 2
Performance comparison for fluids flushing scene.

Method	Pressure Solving		Overall Simulation	
	avg	sum	avg	sum
PCISPH	223.3 ms	1454 s	378.0 ms	2462 s
PP-PCISPH	114.2 ms	918 s	237.9 ms	1480 s
DFSPH	161.0 ms	1036 s	326.0 ms	2099 s
PP-DFSPH	71.5 ms	535 s	205.7 ms	1288 s

This scene contains 369,000 fluid particles in the fluid block with 100,000 high-speed particles to emit later and 448,000 static boundary particles.

Table 3
Performance comparison for high-speed fluids flushing the dragon (Fig. 7).

Method	Pressure solving		Overall simulation	
	avg	sum	avg	sum
PCISPH	80.3 ms	471 s	143.1 ms	840 s
PP-PCISPH	62.7 ms	367 s	133.9 ms	783 s
DFSPH	62.6 ms	330 s	130.6 ms	688 s
PP-DFSPH	29.0 ms	153 s	100.1 ms	529 s

In this scene, 208,000 fluid particles are involved, with 158,000 particles for the dragon and 309,000 particles for the boundary.

6.2.1. High-speed fluids flushing fluid block

We compare the performances like the scene of Fig. 3 but introduce more particles than Fig. 3 in experiments to both high-speed fluids and fluid block. We simulate the scene on PCISPH and DFSPH with or without our scheme to make a comparison. Table 2 shows the detailed performance of the simulation. This scenario illustrates that with our methods, both PCISPH and DFSPH can have a better performance on simulation efficiency. Fig. 6 compares the rendering results with or without our dynamic particle partitioning method, showing that although our PP-SPH method ignores some calculations of low-speed particles, we can still capture sufficient details.

6.2.2. High-speed fluids flushing the dragon

Fig. 7 demonstrates the realistic rendering result of high-speed fluids flushing the dragon model, thus producing a great number of splashes on the dragon and on the ground. Table 3 shows the performance comparison between methods. We can see our method has better performance than the standard SPH methods on pressure solving, but as the number of fluid particles for the scene is at a low level and neighborhood searching is another time-consuming procedure, thus the speed-up on overall performance is a little less than the one on pressure solving.

Fig. 8 demonstrates the number of particles in two groups after processing the particle partitioning procedure. The scene begins with high-speed particle emission, thus most of the particles partitioned into the high-speed group are those emitted particles. At the end of the emission, some of the fast particles become slower and closer to the velocity of normal particles, so the number of high-speed particles shows a trend of decreasing in the later frames. The result of *k*-means clustering shows a similar repeated pattern. When the variation of particle speeds becomes smaller, more particles are prone to the trend of being partitioned into the group of higher velocity, and after the sudden increase on the num-

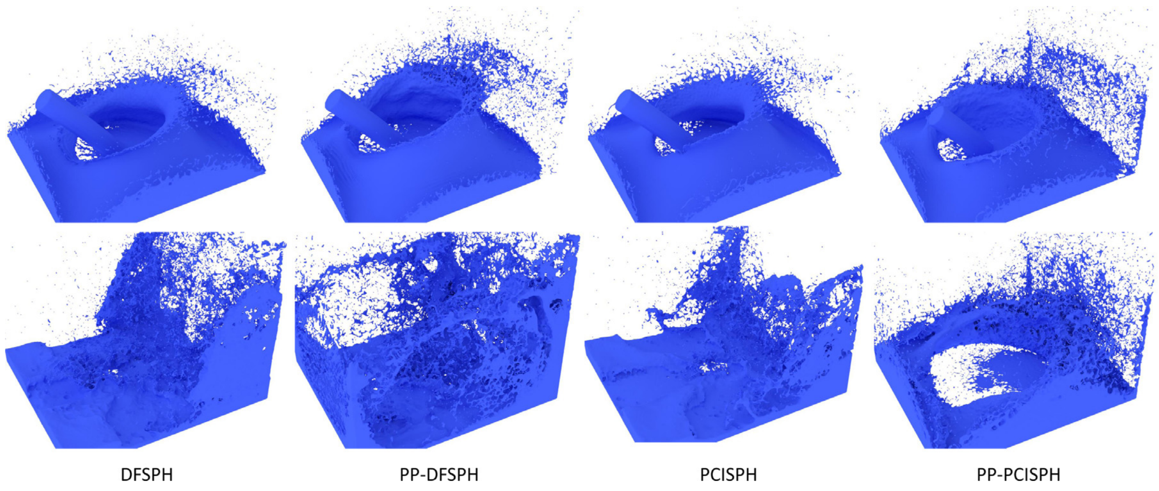

Fig. 6. Comparison for high-speed fluids flushing fluid block with different methods. With our method, we can generate more splashes and preserve more details.

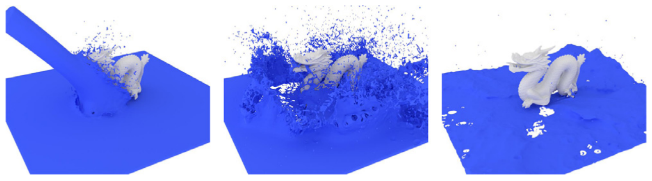

Fig. 7. High-speed fluids flushing the dragon model.

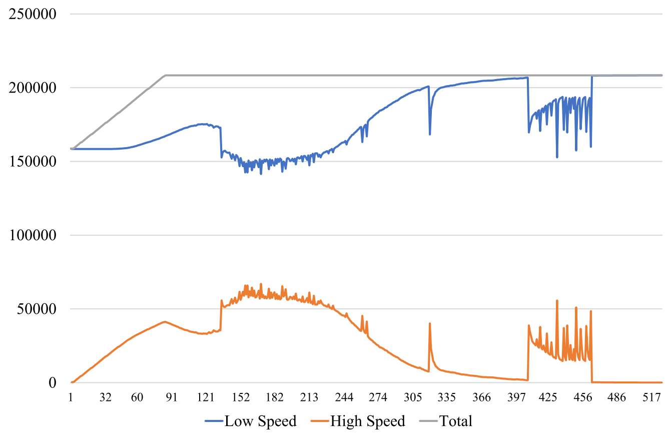

Fig. 8. The Result of particle partitioning procedure for High-speed Fluids Flushing the Dragon (Fig. 7).

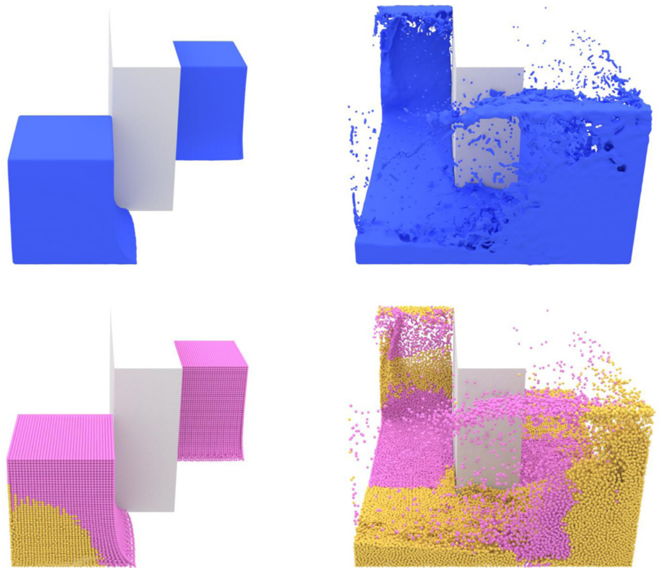

Fig. 9. Double dam break with obstacle. Top row: mesh model views; Bottom row: Particle partitioning view.

ber of high-speed particles, a decreasing trend on the number is presented in the later simulation steps. In some cases, some particles will shift between two groups, thus a fluctuation on the number of particles in each group is presented in the chart.

The result of Figs. 3 and 7 have proved that our methods have a better performance than the standard ones in the scene in which high-speed fluids are hitting the normal-speed fluids or the static boundary objects. But in fact, having high-speed fluids at the beginning of the simulation is not a necessity. Our method will also have a better performance in the scene in which particles can be distinctly partitioned into two groups. It indicates in most scenes where particles will have a large difference in their velocities fields after processing a number of simulation steps. The particle clustering procedure will work fine and our method will show its advantages over the standard methods.

6.3. Comparisons with different parameter settings

We have two more parameters in our method than the standard SPH methods, the maximum mini-steps in a major step $N_{\max}$ and the time step adjustment parameter for low-speed particles ϵ. The parameters can be controlled by users to adjust the simulation results and the efficiency of the simulation. In this section, we conduct a series of experiments to compare the performance when choosing different parameters in the simulation. We choose a scene of double dam breaks in Fig. 9 to compare the performance between different parameter settings.

Table 6 demonstrates the performance with different parameter settings in double dam break scenes. In our method, choosing a larger value for $N_{\max}$ will improve the efficiency of the simulation. In our simulation, we usually choose a number ranging from 2 to 4 for $N_{\max}$. The ϵ is served as a supplement to the time step for low-speed particles. A larger ϵ will decrease the efficiency of the simulation as the computational cost will increase with the time-step size. When ϵ is too large, it will also cause the instability issue as it breaks the rule of the CFL condition. We also notice, the increase of ϵ on PCISPH will have a larger decrease in the effi-

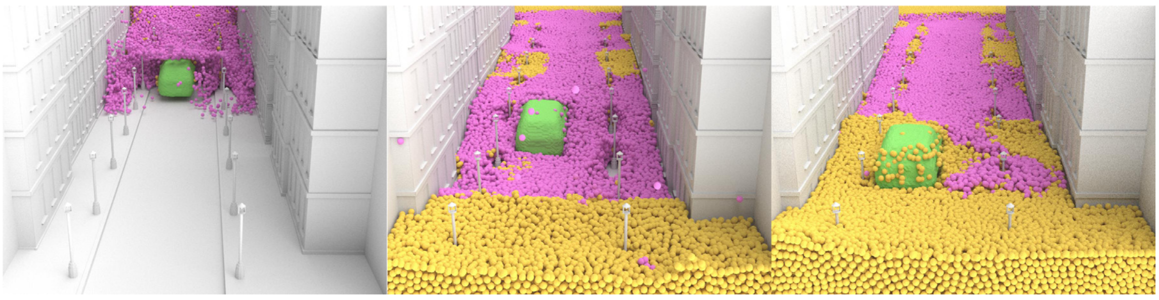

Fig. 10. The result of particle partitioning procedure for rigid body-fluid interaction.

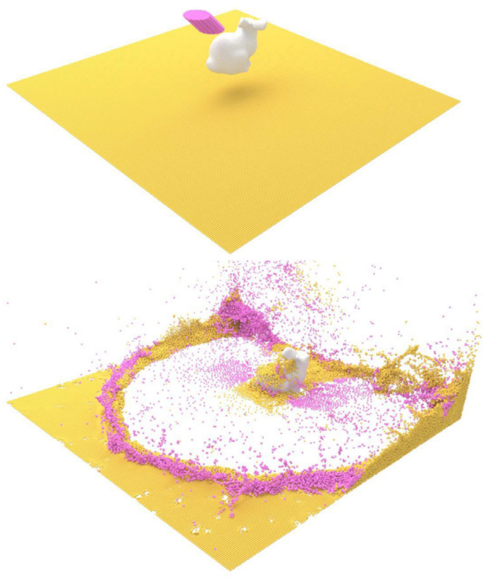

Fig. 11. The result of particle partitioning procedure for soft body-fluid interaction.

ciency comparing to DFSPH because the computational cost of pressure solving in PCISPH grows much faster than DFSPH as time-step increases.

6.4. *Applications for dynamic boundary*

Static obstacles interacting with fluid have been token as static boundaries and achieved good performances. And for solid-fluid interacting scenarios, since the dynamic solid contacts fluid only with a small part of the entire fluid, which means relatively high velocities particles are the minorities [12]. For this condition, our approach has more obvious advantages than traditional methods that need to handle all the particles. To prove that, we replace the static object with shape matching constraint to realize a flexible fluid-solid interactions varying from rigid bodies to deformable objects [42].

Fig. 10 shows a rigid solid-fluid interacting in a flooding-street scene. Fig. 11 illustrates an interaction of fluids flushing a deformable model and Fig. 12 gives a melting phenomenon, in which a melt-able model drops on board and falls into the water to granular melts. Also, Table. 7 illustrates the performances of different SPH models combined with shape matching constraints. From these three experiments we can see that only the fluid particles interacting with dynamic boundaries or on the surfaces tend to be partitioned in high velocity set, so our approach can make a big contribution to this situation and improve a lot of efficiencies.

Furthermore, we have chosen three of all scenarios and computed ratios of $v_{max,\ all}$ to $v_{max,\ low}$ in each frame of the three scenarios, as shown in Fig. 13. Fig. 13(d) shows the comparison of ratios in these three differ-

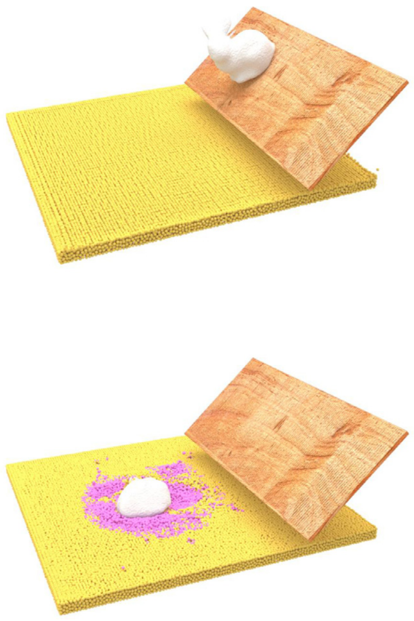

Fig. 12. The result of particle partitioning procedure for melt-able-fluid interaction.

ent scenarios. We can notice that the ratios are roughly between 2 and 10. As is illustrated by the polyline diagram, the ratios vary greatly in every scenario, and different scenarios have completely different ratios. It is difficult to find a threshold that suits all scenarios unless spending a lot of time setting thresholds for every scene.

6.5. *Large-scale scenarios*

We conduct some large-scale scenarios with our scheme to confirm the efficiency and stability of our new method when simulating with a large number of particles.

6.5.1. *Flooding street*

Fig. 15 illustrates a fluid block falling and flushing to the street, hitting the vehicles and street lights. Fluids produce big waves when hitting the vehicles and are divided into two flows when meeting the street lights. Table 4 documents the performance of Figs. 15 and 14 shows the partitioning result. We can notice an increase in the percentage of high-speed particles from Fig. 14. This is because, in the beginning, particles inside the fluid block are with low velocities, while most of the particles near the surface have relatively higher velocities and they are partitioned into the high-speed group.

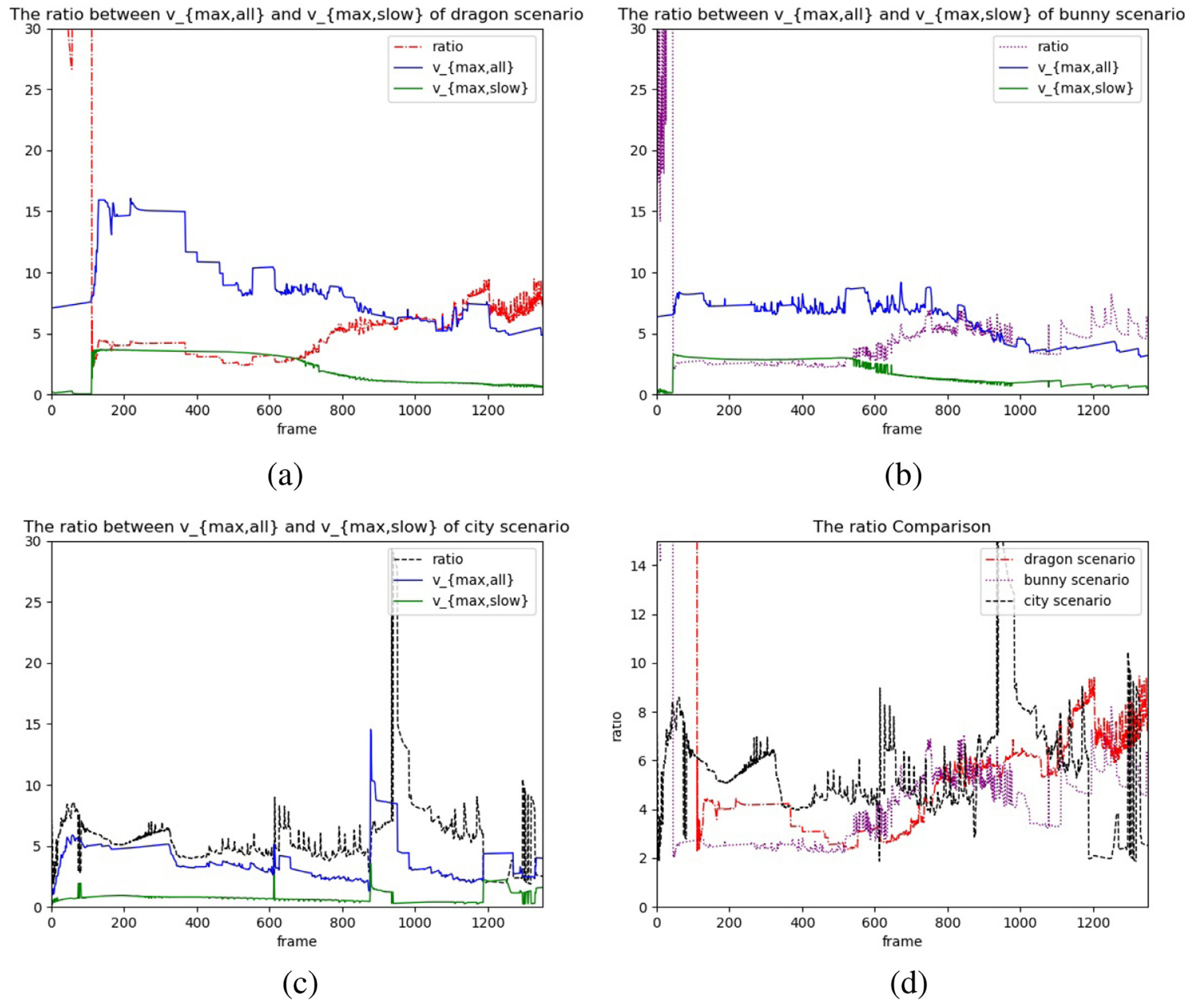

Fig. 13. Comparison of ratios in three different scenarios.

Table 4
Performance comparison of flooding street (Fig. 15).

Method	Num part.	Pressure solving		Overall simulation	
		avg	sum	avg	sum
PCISPH	2720k	1119.54 ms	11,006 s	1499.8 ms	14,787 s
PP-PCISPH		707.4 ms	6780 s	1032.2 ms	9892 s
DFSPH		1202.6 ms	8812 s	1679.2 ms	12,305 s
PP-DFSPH		303.7 ms	3018 s	755.7 ms	7511 s
PCISPH	5330k	2580.9 ms	32,594 s	3871.4 ms	48,892 s
PP-PCISPH		2566.2 ms	25,573 s	3675.2 ms	36,623 s
DFSPH		2956.5 ms	27,433 s	4413.0 ms	40,948 s
PP-DFSPH		841.2 ms	8947 s	2139.7 ms	22,757 s

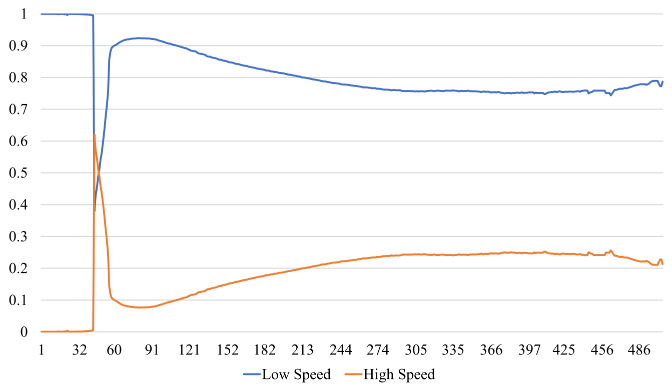

Fig. 14. The result of particle partitioning procedure for flooding street (Fig. 15).

6.5.2. *Flooding city*

Fig. 1 illustrates fluid flooding the city and generating splashes when hitting the buildings and flowing through the roads between buildings. Table 5 documents the performance of Fig. 1.

Both Figs. 15 and 1 have proved that our method is capable of simulating large-scale scenarios with stability and efficiency. We believe, our method should be able to simulate some larger scenarios, yet due to the limitation on the graphics memory at our lab, we are not able to conduct such simulations currently.

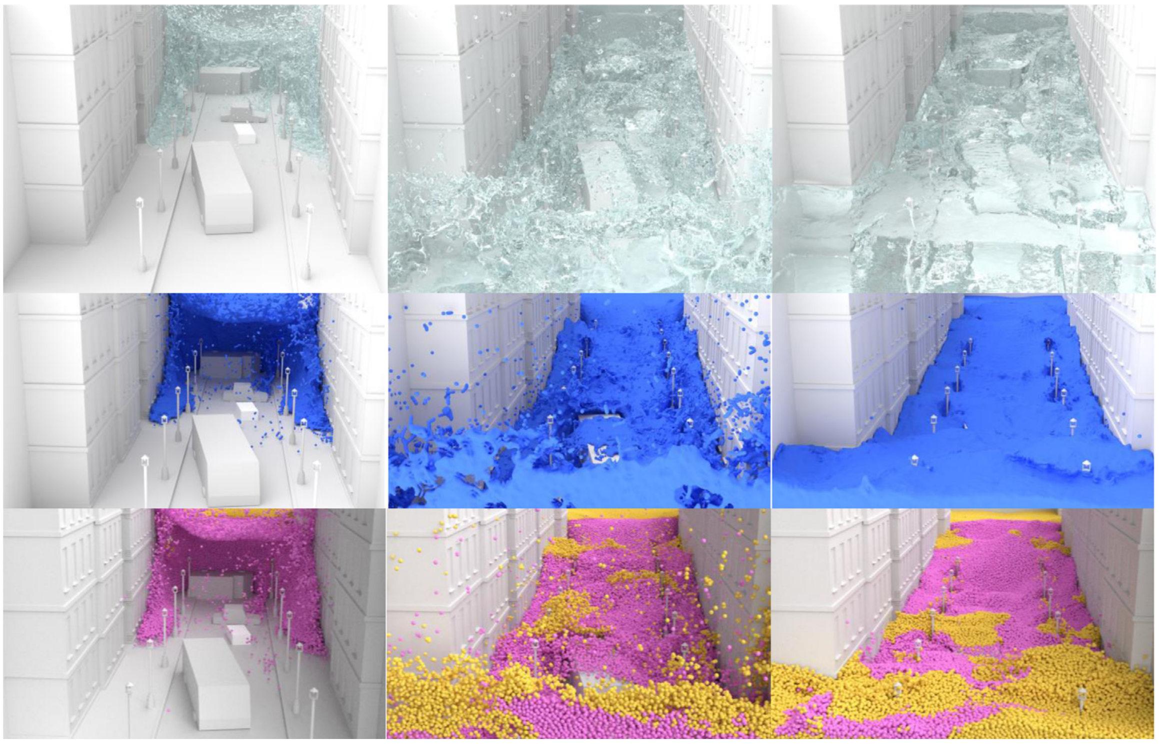

Fig. 15. A fluid block flooding the street. The rendering results are displayed as fluid surfaces and clustered particles.

Table 5
Performance comparison for flooding city (Fig. 1).

Method	Num part.	Pressure solving		Overall simulation	
		avg	sum	avg	sum
PCISPH	7240k	989.4 ms	10,833 s	1699 ms	18,604 s
PP-PCISPH		572.2 ms	6220 s	1195.5 ms	12,995 s
DFSPH		783.7 ms	8519 s	1539.6 ms	16,735 s
PP-DFSPH		378.8 ms	4069 s	1129.2 ms	12,127 s

7. Conclusion

In this paper, we have detailed a novel SPH framework to boost the efficiency and robustness of fluid simulation. In contrast to the previous SPH methods using a static or adaptive yet unified time step for all fluid particles in the simulation, we introduced a two-scale time-step scheme for SPH, which partitions particles into two groups, and enforces different schedules and different time-step sizes for particles of different groups. Simulating particles with different strategies in fluid simulations has resulted in an improvement on computational efficiency, and can guarantee the numerical stability while still producing realistic details.

Combining the experiments in the previous chapters, we compared several popular traditional SPH methods and their derivatives with the PPSPH method proposed in this paper according to the characteristics of several most important aspects of the SPH method. As Table 8 shows, the non-filled star means a half-filled star. The more stars, the better the effect. It can be seen that our method has apparent advantages over existing SPH methods when simulating large-scale phenomena with high-speed particles in the scenes. Our method has apparent advantages over existing SPH methods when simulating large-scale phenomena with high-speed particles in the scenes.

Table 6
Performance comparison for double dam break (Fig. 9).

$N_{\max}$	ϵ	PCISPH pressure avg	PCISPH pressure sum	PCISPH overall avg	PCISPH overall sum	DFSPH pressure avg	DFSPH pressure sum	DFSPH overall avg	DFSPH overall sum
2	1	65.6 ms	159.3 s	112.9 ms	274.2 s	29.1 ms	73.2 s	78.6 ms	197.3 s
	1.3	111.2 ms	256.9 s	160.4 ms	370.6 s	29.3 ms	76.0 s	77.1 ms	196.1 s
	1.5	128.2 ms	302.0 s	177.9 ms	419.2 s	30.7 ms	79.0 s	79.3 ms	204.2 s
3	1	55.9 ms	123.7 s	95.8 ms	211.9 s	24.7 ms	61.7 s	68.8 ms	171.8 s
	1.3	76.0 ms	185.3 s	118.9 ms	289.9 s	26.6 ms	67.4 s	71.2 ms	180.2 s
	1.5	103.2 ms	236.6 s	145.6 ms	333.7 s	26.4 ms	67.4 s	71.3 ms	181.9 s
	2	137.8 ms	307.0 s	176.4 ms	393.1 s	30.2 ms	78.6 s	74.9 ms	195.0 s
4	1	47.3 ms	115.4 s	84.9 ms	207.2 s	28.4 ms	73.5 s	70.7 ms	183.3 s
	1.5	81.5 ms	194.0 s	120.2 ms	285.9 s	29.7 ms	78.5 s	71.0 ms	187.7 s
	2	102.9 ms	245.8 s	139.3 ms	332.9 s	32.7 ms	83.7 s	75.3 ms	192.8 s
	2.5	137.8 ms	339.1 s	170.3 ms	419.2 s	36.0 ms	98.6 s	78.1 ms	214.3 s
Standard		107.0 ms	317.2 s	138.8 ms	411.7 s	48.1 ms	123.7 s	90.3 ms	232.4 s

Table 7
Performance for hhape matching-PPSPH combination experiments.

Scene	*Method*	*NumPart*	Pressure solving avg	Pressure solving sum	Overall simulation avg	Overall simulation sum
Fig. 10	PCISPH	513k	779.6 ms	2058 s	910 ms	2402 s
	PP-PCISPH		481.8 ms	1504 s	612.6 ms	1912 s
	DFSPH		525.4 ms	1523 s	639 ms	1853 s
	PP-DFSPH		227.1 ms	704 s	476.9 ms	1478 s
Scene	*Method*	*NumPart*	Pressure solving avg	Pressure solving sum	Overall simulation avg	Overall simulation sum
Fig. 11	PCISPH	452k	62.0 ms	386 s	104.3 ms	644 s
	PP-PCISPH		40.9 ms	205 s	83.2 ms	417 s
	DFSPH		67.2 ms	329 s	131.0 ms	642 s
	PP-DFSPH		34.7 ms	177 s	81.5 ms	416 s
Scene	*Method*	*NumPart*	Pressure solving avg	Pressure solving sum	Overall simulation avg	Overall simulation sum
Fig. 12	PCISPH	265k	255.3 ms	299 s	284.2 ms	333 s
	PP-PCISPH		203.7 ms	229 s	232.1 ms	260 s
	DFSPH		216.4 ms	237 s	238.0 ms	308 s
	PP-DFSPH		157.5 ms	204 s	193.7 ms	251 s

Table 8
Qualitative evaluation and comparison among existing methods.

Method	Stability	Incompressibility	Time step size	Efficiency
Standard SPH [25]	★	★	★	★
WCSPH [3]	★☆	★☆	☆	★
PCISPH[32]	★★	★★	★★	★★
IISPH [17]	★★★	★★★	★★★	★★★
DFSPH [4]	★★★	★★★	★★★★	★★★★
PP-PCISPH	★★	★★	★★☆	★★★
PP-DFSPH	★★★	★★★	★★★★☆	★★★★★

7.1. Limitation

k-means clustering is the particle partitioning algorithm we had used in the current method, but the result of *k*-means clustering is prone to certain configurations that might be affected by the initial clustering input. We noticed that some particles with intermediate speeds will be rapidly shifting between high-speed and low-speed clusters. Although it won't affect the stability of our simulation, it might slightly reduce the efficiency of our simulation because more particles are partitioned into the high-speed group, thus more computations need to be handled.

7.2. Future works

So far, we have proved the efficiency and stability of our methods with simulations including fluids and dynamic boundary objects. Currently, we partition particles into two groups in our method, but it is more natural to think that it may be better to partition particles more accurately. For example, partitioning fluid particles into more groups, or implementing an adaptive group number for particle partitioning. It might bring more benefits if particles could be partitioned more exactly, as details of the simulation result are more likely to be preserved, and it would help improve the efficiency. It will be one of our goals in the future. We implement the *k*-means clustering algorithm in particle partitioning for its quick convergence and easy implementation in this paper, but there may be some other better clustering methods or variance for our method to resolve problems like particle shifting we have mentioned before. To retain a better and efficient simulation result, the setting of two parameters in our method needs to be determined manually. The choice is more based on our experiences rather than physical theories, and how to automatically find proper parameters is another topic that we wish to focus on next. Meanwhile, we will continue to expand our methods to facilitate more complex applications in the future.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Yang Gao: Conceptualization, Methodology, Visualization, Writing - original draft, Investigation. **Zhong Zheng:** Software, Data curation, Methodology, Writing - original draft. **Jin Li:** Visualization, Validation, Resources, Formal analysis. **Shuai Li:** Project administration, Writing - review & editing. **Aimin Hao:** Project administration, Funding acquisition. **Hong Qin:** Writing - review & editing, Conceptualization, Supervision.

Acknowledgments

This research is supported in part by National Key R&D Program of China (No. 2017YFF0106407), Beijing Advanced Innovation Center for Biomedical Engineering (ZF138G1714), National Natural Science Foundation of China (Nos. 61672077 and 61532002) and National Science Foundation of USA (IIS-1715985 and IIS-1812606).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi;10.1016/j.gmod.2020.101061

References

[1] B. Adams, M. Pauly, R. Keiser, L.J. Guibas, Adaptively sampled particle fluids, in: ACM Transactions on Graphics (TOG), 26, ACM, 2007, p. 48.
[2] C. Batty, R. Bridson, Accurate viscous free surfaces for buckling, coiling, and rotating liquids, in: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Eurographics Association, 2008, pp. 219–228.
[3] M. Becker, M. Teschner, Weakly compressible SPH for free surface flows, in: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Eurographics Association, 2007, pp. 209–217.
[4] J. Bender, D. Koschier, Divergence-free SPH for incompressible and viscous fluids, IEEE Trans. Vis. Comput. Graph. 23 (3) (2017) 1193–1206.
[5] L. Boyd, R. Bridson, Multiflip for energetic two-phase fluid simulation, ACM Trans. Graph. (TOG) 31 (2) (2012) 1–12.
[6] N. Chentanez, M. Müller, Real-time Eulerian water simulation using a restricted tall cell grid, in: ACM Transactions on Graphics (TOG), 30, ACM, 2011, p. 82.
[7] M. Chu, N. Thuerey, Data-driven synthesis of smoke flows with CNN-based feature descriptors, ACM Trans. Graph. (TOG) 36 (4) (2017) 69.
[8] N. Foster, R. Fedkiw, Practical animation of liquids, in: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, ACM, 2001, pp. 23–30.
[9] Y. Gao, S. Li, H. Qin, A. Hao, A novel fluid-solid coupling framework integrating flip and shape matching methods, in: Proceedings of the Computer Graphics International Conference, ACM, 2017, p. 11.
[10] Y. Gao, S. Li, L. Yang, H. Qin, A. Hao, An efficient heat-based model for solid-liquid-gas phase transition and dynamic interaction, Graph. Models 94 (2017) 14–24.
[11] D. Gerszewski, A.W. Bargteil, Physics-based animation of large-scale splashing liquids, ACM Trans. Graph. (TOG) 32 (6) (2013) 1–6.
[12] C. Gissler, A. Peer, S. Band, J. Bender, M. Teschner, Interlinked SPH pressure solvers for strong fluid-rigid coupling, ACM Trans. Graph. (TOG) 38 (1) (2019) 1–13.
[13] Y. Guo, X. Liu, X. Xu, A unified detail-preserving liquid simulation by two-phase lattice Boltzmann modeling, IEEE Trans. Vis. Comput. Graph. 23 (5) (2017) 1479–1491.
[14] J.A. Hartigan, M.A. Wong, Algorithm as 136: a k-means clustering algorithm, J. R. Stat. Soc. Ser. C (Appl. Stat.) 28 (1) (1979) 100–108.
[15] X. He, N. Liu, S. Li, H. Wang, G. Wang, Local poisson SPH for viscous incompressible fluids, in: Computer Graphics Forum, 31, Wiley Online Library, 2012, pp. 1948–1958.
[16] Y. Hu, Y. Fang, An asynchronous material point method, in: ACM SIGGRAPH 2017 Posters, ACM, 2017, p. 60.
[17] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, M. Teschner, Implicit incompressible SPH, IEEE Trans. Vis. Comput. Graph. 20 (3) (2014) 426–435.
[18] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, M. Teschner, SPH fluids in computer graphics, Eurographics 2014 - State of the Art Reports, The Eurographics Association, 2014.
[19] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, M. Gross, Data-driven fluid simulations using regression forests, ACM Trans. Graph. (TOG) 34 (6) (2015) 1–9.

[20] M. Macklin, M. Müller, Position based fluids, ACM Trans. Graph. (TOG) 32 (4) (2013) 104.
[21] M. Macklin, M. Müller, N. Chentanez, T.-Y. Kim, Unified particle physics for real–time applications, ACM Trans. Graph. (TOG) 33 (4) (2014) 153.
[22] M.-L. Eckert, K. Um, N. Thuerey, T.-Y. Kim, ScalarFlow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning, ACM Trans. Graph. 38 (6) (2019) 239.
[23] J. Monaghan, On the problem of penetration in particle methods, J. Comput. Phys. 82 (1) (1989) 1–15.
[24] J.J. Monaghan, Smoothed particle hydrodynamics, Annu. Rev. Astron. Astrophys. 30 (1) (1992) 543–574.
[25] M. Müller, D. Charypar, M. Gross, Particle-based fluid simulation for interactive applications, in: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer Animation, Eurographics Association, 2003, pp. 154–159.
[26] A. Paiva, F. Petronetto, T. Lewiner, G. Tavares, Particle-based viscoplastic fluid/solid simulation, Comput. Aided Des. 41 (4) (2009) 306–314.
[27] A. Peer, M. Ihmsen, J. Cornelis, M. Teschner, An implicit viscosity formulation for SPH fluids, ACM Trans. Graph. (TOG) 34 (4) (2015) 114.
[28] A. Peer, M. Teschner, Prescribed velocity gradients for highly viscous SPH fluids with vorticity diffusion, IEEE Trans. Vis. Comput. Graph. 23 (12) (2017) 2656–2662.
[29] K. Raveendran, C. Wojtan, N. Thuerey, G. Turk, Blending liquids, ACM Trans. Graph. 33 (4) (2014) 1–10.
[30] S. Reinhardt, M. Huber, B. Eberhardt, D. Weiskopf, Fully asynchronous SPH simulation, in: B. Thomaszewski, K. Yin, R. Narain (Eds.), Eurographics/ ACM SIGGRAPH Symposium on Computer Animation, ACM, 2017, doi:10.1145/3099564.3099571.
[31] H. Schechter, R. Bridson, Ghost SPH for animating water, ACM Trans. Graph. (TOG) 31 (4) (2012) 61.
[32] B. Solenthaler, R. Pajarola, Predictive-corrective incompressible SPH, in: ACM Transactions on Graphics (TOG), 28, ACM, 2009, p. 40.
[33] D. Stora, P.-O. Agliati, M.-P. Cani, F. Neyret, J.-D. Gascuel, Animating lava flows, in: Graphics Interface (GI'99) Proceedings, 1999, pp. 203–210.
[34] T. Takahashi, Y. Dobashi, I. Fujishiro, T. Nishita, M.C. Lin, Implicit formulation for SPH-based viscous fluids, in: Computer Graphics Forum, 34, Wiley Online Library, 2015, pp. 493–502.
[35] B. Thomaszewski, S. Pabst, W. Straßer, Asynchronous cloth simulation, Computer Graphics International, 2, 2008.
[36] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin, Accelerating Eulerian fluid simulation with convolutional networks, in: Proceedings of the 34th International Conference on Machine Learning, 2017, pp. 3424–3433.
[37] K. Um, X. Hu, N. Thuerey, Liquid splash modeling with neural networks, Comput. Graph. Forum 37 (8) (2018) 171–182.
[38] C. Wang, Q. Zhang, H. Xiao, Q. Shen, Simulation of multiple fluids with solid-liquid phase transition, Comput. Animat. Virtual Worlds 23 (3–4) (2012) 279–289.
[39] M. Weiler, K. Dan, M. Brand, J. Bender, A physically consistent implicit viscosity solver for SPH fluids, Comput. Graph. Forum 37 (2) (2018) 145–155.
[40] Y. Xie, E. Franz, M. Chu, N. Thuerey, TempoGAN: a temporally coherent, volumetric GAN for super-resolution fluid flow, arXiv:1801.09710 (2018).
[41] C. Yang, X. Yang, X. Xiao, Data-driven projection method in fluid simulation, Comput. Animat. Virtual Worlds 27 (3–4) (2016) 415–424.
[42] G. Yang, L. Shuai, Q. Hong, Y. Xu, A. Hao, An efficient flip and shape matching coupled method for fluidsolid and two-phase fluid simulations, Vis. Comput. (4) (2018) 1–13.
[43] L. Yang, S. Li, A. Hao, H. Qin, Hybrid particlegrid modeling for multiscale droplet/spray simulation, Comput. Graph. Forum 33 (7) (2015) 199–208.
[44] Y. Zhu, R. Bridson, Animating sand as a fluid, ACM Trans. Graph. (TOG) 24 (3) (2005) 965–972.