Fully Asynchronous SPH Simulation

Seminar: Current Topics in Fluid Animation

Yinglun Liu

June 19, 2019

1 Motivation

Over the years, the adoption of smoothed particle hydrodynamics(SPH) has developed to become the common pratice when simulating fluids in various scenes. In non-iterative approaches, the physical properties of a fluid particle is constantly influenced by its immediate neighborhood and is thereby updated in each time step following a procedure that helps compute the respective components in the Navier-Stokes equation. In these equations, the computation of new attributes of the current particle requires several times the access to information from its surrounding neighbors. Intuitively, a natural practice would be to, as did many non-iterative SPH solvers, perform global updates to all particles using a single uniform step size. While iterative SPH solvers nowadays generally yield better performances, traditional non-iterative approaches do not seem to lessen in popularity due to the fact that they are easy to implement and suitable for less turbulent fluids (see Fig. 1).

Be that as it may, a core defect of non-iterative SPH solvers may lie in the lack of efficiency. On one hand, to enforce a constantly negligible density deviation within the fluid, large stiffness parameters are adopted to generate high pressures. Such selection

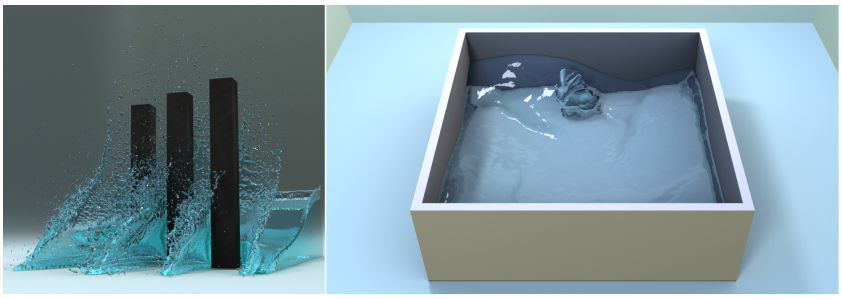

Figure 1: Animated scenes of dam break and radial flow demonstrate that non-iterative SPH solvers are capable of generating fluid scenes with high visual authenticity [RHEW17].

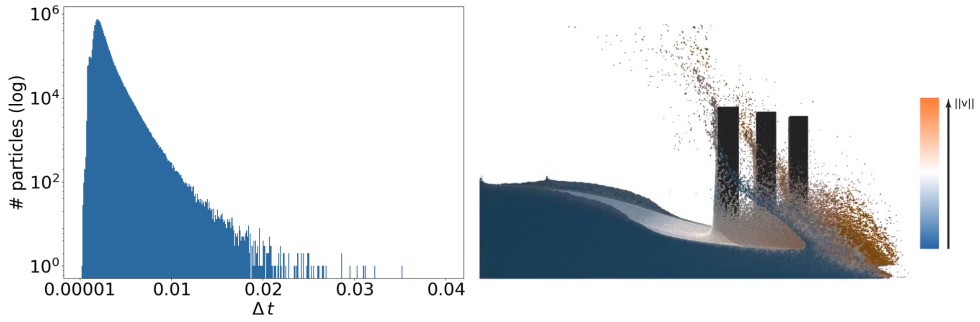

Figure 2: For fluid simulation based on SPH, the magnitude of velocity differs substantially across particles from different regions (right). As a result, the maximum possible step size for each individual particle could diverge to cover up to multiple magnitudes (left) [RHEW17].

of parameters demands smaller time steps and, in turn, a greater number of global iterations for stable and correct particle interations. This can lead to tremendously higher overhead for the generation of visually realistic animation, where tens of millions of particles are included to allow for as fine-grained details as possible. On the other hand, the paper made the observation that, while smaller time steps are essential in heavily interacting regions to guarantee stable simulation, in many less complex parts of the fluid the particles could do well with a larger time step (see Fig. 2). That is, the maximum possible step size for a particle varies substantially across different regions of the simulation, and it makes less sense to set one global step size for all particles, since it would inevitably be limited by the single particle with the strictest time constraint and lead to much waste of computational power. Previous works that attempted to tackle this aspect of the problem had to introduce some global synchronization barriers that improves consistency but somehow shadows the boost on performance due to waiting threads within parallel execution.

To cope with this, the paper proposes a novel method for the time integration of particles, in which each particle is assigned an individual step size and the whole simulation is carried out fully asynchronous. By handling each particle individually throughout the simulation, the solver expends less computational effort on smoother regions and considerably cut down the overall time consumption. To ensure consistency, the neighborhood is reconstructed at the current time stamp each time a particle is processed. To demonstrate the strength of the proposed method, the paper further proposes a multi-queue parallelization to such fully asynchronous integration procedure. Comparative experiments against related works were conducted under diverse conditions and environments to prove the efficacious enhancement on performance.

2 Related Work

Since the contribution of this paper focuses on the temporal adaptivity of SPH-based fluid simulations (i.e. how the step size of SPH integration models could be adapted to enhance the performance of the simulation) and how the proposed method could be parallelized, we discuss related works in these two directions separately.

2.1 Temporal Adaptivity

Extensive studies have been conducted on the temporal adaptivity of SPH models:

- For synchronous SPH simulation, temporal adaptivity could be exploited as in [DG96]. For each global iteration the uniform step size is determined by the particle with the strictest CFL constraint. This scheme is also adopted by [BET+10], who added an additional acceleration constraint as the second term to take into consideration the impact of large forces.
- [GB14] introduces a regionally synchronous integration model. They divide all particles into separate rigions depending on their maximum possible step size and fix a uniform step size for each region that is a multiple of the user-defined base value.
- [DG96] explored the possibility to apply individual times steps. For each particle the user-defined simulation step size is divided by powers of two to the extent such that the CFL condition is met. Forces are evaluated only at individual time steps to facilitate gains in performance. [BWH+18] follows this idea and proposes a semi-synchronous time stepping scheme. They introduce the concept of a virtual system time line that advances with the smallest individual step size of all particles. Particles with an individual step size that exceeds the gap between its own time stamp and the system checkpoint are deactivated and only propagated forward using linear interpolation. Those that are active are instead evaluated in the same procedure as in the fully synchronous setting. Such a stepping strategy still introduces a great number of global synchronization barriers although the overall operations are cut down to some extent.

2.2 Parallelism in SPH Simulation

Speeding up SPH simulations can also be achieved by exploiting parallelism:

- For neighborhood search and access to particle properties, the community seems to have reached the consensus to utilize a uniform grid to facilitate concurrent queries of neighborhood information [IOS+14].
- For synchronous non-iterative SPH simulation, simple loop parallelization could be adopted, since several for-loops are responsible for computing the attributes of the particles in each global step.

- For [GB14], parallel computation should be straightforward to implement as for each region the step size is fixed. However, the solver could suffer from unbalanced load and the waiting threads since at the edge zones of the regions synchronization becomes inevitable.
- For [BWH+18], since in each global synchronization barrier the solver performs either full integration or linear interpolation on all particles, the parallelization scheme bears little difference to the fully synchronous setting.

3 Non-iterative SPH

Governed by the Navier-Stokes equation (1), non-iterative SPH performs interpolation over the neighborhood of a particle to calculate its attributes and ultimately the forces it incurs within one time step of the simulation. The fluid attribute A_i of particle i at position $\boldsymbol{x}_i$ is computed as the weighted sum of A_j over all neighboring particles j, with the weights given by a user-defined kernel function W that satisfies the following conditions:

- The function has an integral value of one over its acting domain.
- The function approximates the *Dirac-*δ distribution as its smoothing length approaches zero.
- The function is non-negative.
- The function is symmetric.
- The function is of compact support (i.e. the function value remains zero beyond its smoothing length).

After all forces are computed, time integration methods like symplectic Euler or leap frog is carried out to update particle position and speed. Denoting with p the pressure, $\boldsymbol{v}$ the velocity and ρ the density of a particle, a typical formulation of the Navier-Stokes equation models the acceleration of a particle as

$$\frac{\delta \boldsymbol{v}}{\delta t} = -\frac{\nabla p}{\rho} + \nu \nabla^2 \boldsymbol{v} + \frac{\boldsymbol{F}^{external}}{\rho}, \tag{1}$$

where ν is the kinematic viscocity of the liquid and the three terms on the right hand side of the equation describes per unit volume the impact of pressure forces, viscosity forces and external forces respectively.

3.1 SPH Procedure with splitting

The paper adopts here the splitting strategy as described by [IOS+14], where for each update step the advection forces $\boldsymbol{F}^{advection}$ are separately computed and then used to calculate an intermediate advection velocity $\boldsymbol{v}^*$. An advection density ρ^* is introduced to

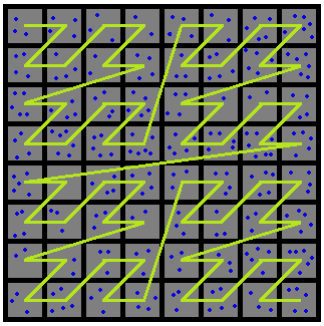

Figure 3: A Z-order curve in 2-dimensional space.

represent the expected density at the end of this time step as a result of the divergence in the velocity field assuming no compensating pressure force is generated. pressures p and pressure forces $\boldsymbol{F}^{pressure}$ are afterwards handled conventionally. This updating scheme forces the solver to implicitly consider the impact of advection forces before generating pressure forces to counteract the deviation of density and therefore stablizes the simulation.

Conventionally, a single global update step with the concept of spitting applied proceeds as in Alg. 1. To interpolate particle properties a neighborhood search is conducted at each time step to fetch the set of neighbors for each particle. Since such operations can be computationally expensive, spatial data structures(e.g. a uniform grid) that supports parallel operations are usually constructed at the beginning to accelerate the process. At each global step, the grid cells are queried and updated to efficiently maintain neighborhood information. To further minimize time consumed by inevitable query operations at each time step, particles are sorted in accordance with their spatial cell. In this way, spatially adjacent particles are stored on adjacent memory slots, enhancing the cache-hit rates during the simulation. Here, the paper chooses Z-order curve (see Fig. 3), a commonly used sorting function that effectively preserves spatial locality due to its fractal block structure, for particle indexing.

The advection force incurred by a particle consists of a viscosity component and some external components. Viscosity forces counteract divergence of the velocity field and is hence proportional to the second-order spatial derivative of velocity. But since second-order derivatives are tricky to compute, the approximation form proposed originally by [Mon92] is prefered instead. Therefore the viscosity component acting on particle i is computed as

$$\boldsymbol{F}_i^{viscosity} = 2m_i\nu \sum_j \frac{m_j}{\rho_j}\boldsymbol{v}_{ij}\frac{\boldsymbol{x}_{ij}\cdot\nabla W_{ij}}{\boldsymbol{x}_{ij}\cdot\boldsymbol{x}_{ij}+0.01h^2}, \tag{2}$$

where m_i denotes the mass of particle i, $\boldsymbol{v}_{ij} = \boldsymbol{v}_i - \boldsymbol{v}_j$ and $\boldsymbol{x}_{ij} = \boldsymbol{x}_i - \boldsymbol{x}_j$. Since modeling boundary interations and surface tension is not a core theme of this paper, the paper directly adopts the method of [AIA+12] and [HRWE15] to handle these two aspects of the advection force.

Algorithm 1 : One global step with splitting [RHEW17]

Function `GlobalStep()`**:**
- **for** *each particle i* **do**
  - find neighbors j
- **end**
- **for** *each particle i* **do**
  - compute advection force $\boldsymbol{F}_i^* = \boldsymbol{F}_i^{viscosity} + \boldsymbol{F}_i^{ext}$
  - compute advection velocity $\boldsymbol{v}_i^*$ using $\boldsymbol{F}_i^*$
- **end**
- **for** *each particle i* **do**
  - compute advection density ρ_i^*
  - compute pressure p_i
- **end**
- **for** *each particle i* **do**
  - compute pressure force $\boldsymbol{F}_i^*$
  - compute new particle velocity $\boldsymbol{v}_i(t + \delta t)$
  - compute new particle position $\boldsymbol{x}_i(t + \delta t)$
- **end**

Based on the advection forces, intermediate advection velocity is computed as

$$\boldsymbol{v}_i^* = \boldsymbol{v}_i(t) + \delta t \frac{\boldsymbol{F}_i^{advection}}{m_i}. \tag{3}$$

This equation can be perceived as an intermediate velocity update using symplectic Euler without considering the impact of pressure forces. The first-order spatial derivative of velocity reflects the divergence of the velocity field and is used to compute the advection density. Again, the paper chooses the robust variant presented by [Mon92] and computes the advection density via

$$\rho_i^* = \sum_j m_j W_{ij} + \delta t_i \sum_j (\boldsymbol{v}_i^* - \boldsymbol{v}_j^*) \cdot \nabla W_{ij}, \tag{4}$$

where the first term describes the static density at particle position $\boldsymbol{x}_i$ and the second term describes the density change during step δt_i as a consequence of diverging velocity field.

Having obtained the advection densities of the particles, pressure at particle position $\boldsymbol{x}_i$ is computed as

$$p_i = k(\rho_i^* - \rho_0), \tag{5}$$

where ρ_0 is the system's reference density and k is the stiffness parameter. Negative pressures are clamped to zero as in [ICS+13] to prevent the water splashes from being

absorbed as a result of the induced cohesive effect. Finally, the respective pressure forces are calculated using the momentum-preserving variant:

$$\boldsymbol{F}_i^{pressure} = -m_i \sum_j m_j \left(\frac{p_i}{(\rho_i^*)^2} + \frac{p_j}{(\rho_j^*)^2} \right) \nabla W_{ij}. \tag{6}$$

This pressure force is then solely responsible for the explicit velocity and position update of the particle, which we discuss in the next subsection.

3.2 Time Integration

Despite the fact that multiple common numerical time integration schemes would fit well into the framework of non-iterative SPH solvers, the paper employs a modified version of symplectic Euler:

$$\begin{aligned} \boldsymbol{v}_i(t+\delta t) &= \boldsymbol{v}_i(t) + \delta t \frac{\boldsymbol{F}_i^{pressure}(t)}{m_i} \\ \boldsymbol{x}_i(t+\delta t) &= \boldsymbol{x}_i(t) + \delta t \boldsymbol{v}_i(t+\delta t) + \frac{\delta t^2}{2} \frac{\boldsymbol{F}_i^{pressure}(t)}{m_i}. \end{aligned} \tag{7}$$

Here δt denotes the individual step size of particle i. In addition to a first-order Taylor expansion, a second-order term is added here to the position updating equation to facilitate stabler behavior when applying larger time steps. For numerical stability and correct behavior of the particles, the size of each global time step has to be chosen in a way such that it fulfills the CFL condition. Intuitively, CFL condition enforces that a particle does not travel in one time step a distance that exceeds the smoothing length of the interpolating kernel, which is typically set to be twice as large as the initial particle spacing r_i. Such a constraint could guarantee that a particle does not leap over its spatial neighbors and thus fail to interact properly with them. Similarly, [BET+10] introduces a constraint on acceleration of the particle that targets the same outcome. Putting these two constraints together, the maximum step size for an individual particle is given by:

$$\delta t_i = min(\lambda_v \frac{2r_i}{|\boldsymbol{v}_i|}, \lambda_F \sqrt{\frac{2r_i}{\boldsymbol{F}_i^{pressure}/m_i}}), \tag{8}$$

where λ_v and λ_F scales the two terms respectively. In previous approaches where a globally adaptive stepping scheme is used, the system-wide step size is naturally chosen to be minimum of Eq. (8) over all particles.

4 Fully Asynchronous SPH

As explained in previous sections, conventional approaches of fixed or globally adaptive time stepping squander computational resources on particles that could have been

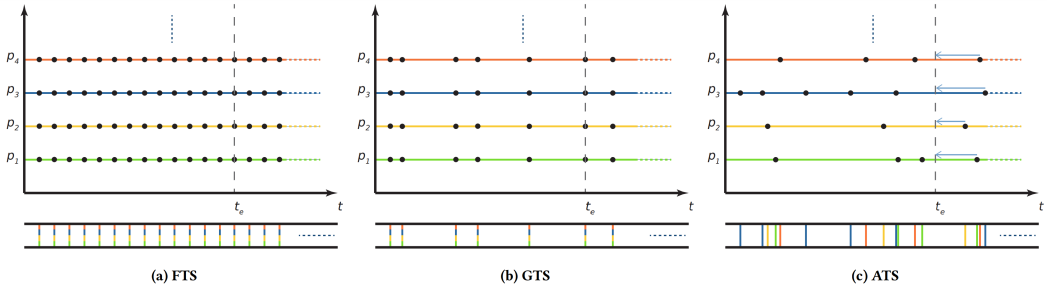

Figure 4: Illustration of various time stepping methods: (a) fixed time stepping (FTS), (b) globally adaptive time stepping (GTS), and (c) asynchronous time stepping (ATS). Note that with FTS and GTS all particles are updated simultaneously at each time step. With ATS each particle freely advances in time at its own pace, minimizing the overall computational expenditure. Only virtual barriers are need for exporting the outcome of the simulation. A priority queue is required to enforce a strict particle processing order for attribute consistency. [RHEW17].

integrated with larger time steps. Former works that try to mitigate the waste on computational power suffer from waiting threads since they introduce multiple times more synchronization barriers within the duration of the simulation. Therefore, the paper proposes to assign to each individual particle a dedicated step size determined by Eq. (8) and avoid any global synchronization barrier. Only a virtual export barrier is employed from time to time to fetch particle states for rendering purposes. A comparative illustration of the proposed method is given in Fig. 4.

4.1 Sequential Execution of Asynchronous Simulation

To achieve such asynchronous SPH simulation, the solver has to constantly ensure that the current particle i has immediate access to the attributes of its neighboring particles at time stamp t_i. Since all particles advance through time independently, this can only be achieved if the attribute A_j of any neighbor j can be temporally interpolated as

$$\delta A_j(t_i) = A_j(t_j) + (t_i - t_j)\frac{\delta A_j(t_j)}{\delta t}. \tag{9}$$

Fortunately, by employing the splitting concept, the non-iterative SPH integration already provides a feasible way to conduct such reconstruction efficiently.

Similar to Alg. 1, one step of integration for an individual particle has the following dependencies:

- The viscosity force $\boldsymbol{F}_i^{viscosity}(t_i)$ incurred by particle i is dependent on the positions $\boldsymbol{x}_j(t_i)$ and velocities $\boldsymbol{v}_j(t_i)$ of neighboring particles j.
- The advection density ρ_i^* of particle i is dependent on the advection velocities $\boldsymbol{v}_j^*$ of neighboring particles j.

- The pressure force $\boldsymbol{F}_i^{pressure}(t_i)$ incurred by particle i is dependent on the advection densities ρ_j^* and pressures p_j of neighboring particles j.

Since velocity and position can be directly pushed forth or back in time under the symplectic Euler scheme, velocities $\boldsymbol{v}_j$ and positions $\boldsymbol{x}_j$ of neighboring particles j are reconstructed at the current time stamp t_i to facilitate computation of viscosity force of the current particle. Since the second term in Eq. (4) approximates the temporal derivative of density, advection densities ρ_j^* of neighboring particles j can be direcly pushed forth or back in time to facilitate computation of pressure force. The pressure p_j of neighboring particles at the current time stamp t_i can be computed as usual since they do not induce a high expense. Since the reconstruction of the advection velocities $\boldsymbol{v}_j^*$ of neighboring particles could be costly, we may assume that the paper replaces $\boldsymbol{v}_j^*(t_i)$ with $\boldsymbol{v}_j(t_i)$ to cut down the total amount of operations for neighborhood reconstruction in a single particle state update. Alg. 2 describes the outline of one individual simulation step. An illustration of the neighborhood reconstruction scheme is given in Fig. 5.

Algorithm 2 : One individual step in ATS [RHEW17]

Function `IndividualStep(`*particle i*`):`

determine maximum possible step size δt_i

reconstruct velocities $\boldsymbol{v}_j(t_i)$ and positions $\boldsymbol{x}_j(t_i)$ of neighbors j at t_i

compute viscosity force $\boldsymbol{F}_i^{viscosity}(t_i)$ incurred by particle i using Eq. (2)
compute external forces $\boldsymbol{F}_i^{ext}(t_i)$ incurred by particle i
compute advection force $\boldsymbol{F}_i^*(t_i)$ incurred by particle i
compute advection velocity $\boldsymbol{v}_i^*(t_i)$ of particle i using Eq. (3)
compute advection density ρ_i^* of particle i using Eq. (4)
compute pressure p_i of particle i using Eq. (5)

reconstruct advection densities ρ_j^* and pressures p_j of neighbors j at t_i

compute pressure force $\boldsymbol{F}_i^{pressure}(t_i)$ incurred by particle i using Eq. (6)

integrate particle i over time using δt_i

To guarantee stable and correct behavior of the algorithm, the paper stipulates empirically that the attributes of neighboring particles should only be traced back in time. In other words, a particle is only to be processed if its entire neighborhood is more advanced in time than itself. To explicitly enforce such strict processing order, a priority queue storing only particle indices is introduced to constantly arange the particles in chronological order. Specificaly, until the next virtual export barrier is reached, the solver repeatedly dequeues the particle at the top of queue, performs one individual step for the particle and then inserts its index back into the queue such that the chronologi-

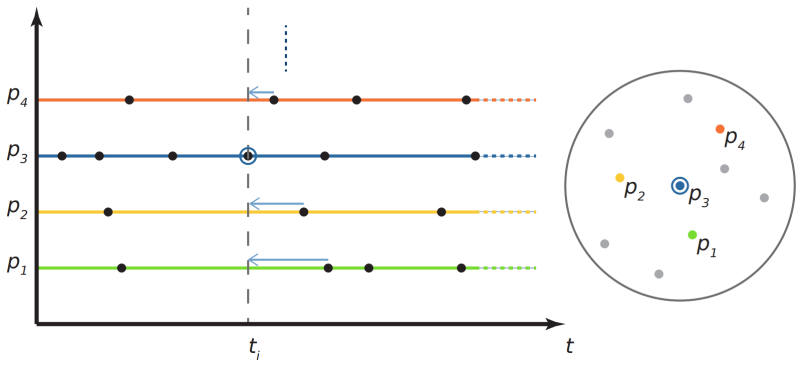

Figure 5: Illustration of the neighborhood reconstruction scheme: attributes of neighbors of particle p_3 are traced back in time to support the computations at the current time step. A particle is processed only when all its neighbors are already temporally ahead of itself. In this particular scenario particle p_4 reaches the head of the priority queue after particle p_3 finishes its individual step [RHEW17].

cal order within the queue is sustained. After the last particle reaches the next virtual export barrier t_{exp}, the attributes of all particles that are demanded by the rendering process are reconstructed at t_{exp} using the same procedure as stated above.

4.2 Parallelization

A parallel implementation of the aforementioned time stepping scheme is not straightforward given the restriction on processing order. Naively spawning multiple queues and handling them independently would lead to violation of the ordinal constraint. Adopting simple synchronization strategies could cause multiple threads to be stuck at each other when several particles have neighbors in other queues that lag behind in time, which could force their belonging queues to hang up until the ordinal constraint is fulfilled. To cope with this undesirably inherent character of the devised approach, the paper introduces per queue a waiting list as the buffer area. When the solver of a queue runs into a particle that violates the ordinal constraint, this particle is removed and pushed into the waiting list (see Fig. 6). The solver then goes on to process the particle at the top of the queue until a predefined interval is reached and all particles in the waiting list are reinserted into the queue. Alg. 3 describes the running process of a priority queue.

The proposed parallelization scheme presupposes a few conditions to ensure its efficiency. To justify the feasibility of the approach, we need to investigate a few implementation details with discretion.

Step size clipping Despite global synchronization barriers are avoided throughout the simulation, in practice it still helps to clip the invididual time steps to be a multiple of 0.5 ms as in [GB14]: Chances are high that spatially adjacent particles tend to be assigned

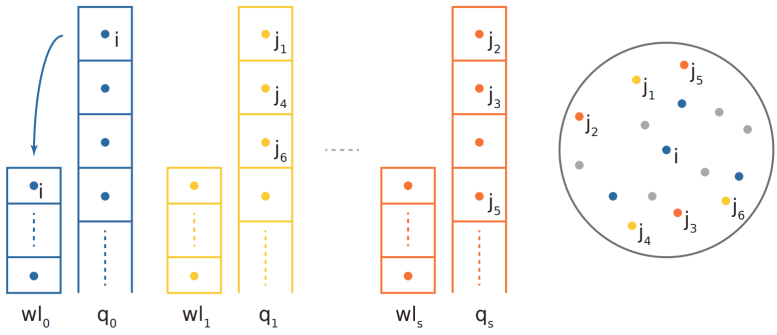

Figure 6: Illustration of the waiting list: the strict particle processing order could be violated when using multiple queues for parallel computation. If a particle at the head of queue q_0 has a neighbor in another queue that is less advanced in time, the solver moves it from q_0 and stores it on the accompanying waiting list wl_0 [RHEW17].

the same step size. When a particle has a neighbor that is equally advanced in time thanks to this clipping operation, the reconstruction of this particular neighbor could be omitted. This could be a huge drop in amount operations for interaction-intensive regions.

Maintenance of neighborhood information To grant efficient query and update of particle neighbors, a uniform grid storing the cells in a one-dimensional array using Z-curve indexing is employed. Since the full construction of such spatial data structure induces a high computational expenditure, a global update of all grid cells is only carried out at each virtual export barrier. To ensure the correctness of a usual neighborhood search implementation, the particles have to be monitored each time they advance in space and time. If a particle changes cell after one individual step, a reference to the particle is instantly added to the new cell without removing the old reference. This is to deal with cases where one neighbor j of a particle i is no longer in i's vicinity after advancing but by the current time stamp t_i it still was and should contribute to particle i's attribute computation. Since the kernel function is of compact support, such cell update strategy does not lead to false neighbors and only induces a small extra cost.

Queue splitting Since a Z-curve indexing of the particle arrays is employed, spatially close particles are assigned to the same queues by simply splitting the queues by particle index. Such property is desirable in the sense that if a particle has too many neighbors that appear in other queues it would inevitably be frequently put into the waiting list. Nevertheless as particles travel during the simulation, such spatial separation is detrimented. The paper proposes to reinitialize the queues after several virtual export barriers to improve spatial clustering among the particles. Empirically, a reinitialization of the queues is done every half of a second, which serves as a good trade-off between

Algorithm 3 : Running process of a priority queue [RHEW17]

```
Function RunQueue(queue q, time stamp t_exp):
    Initialize waiting list wl
    Initialize step count cnt
    while t_{q.top()} < t_exp do
        i = q.pop()
        find all neighbors j of particle i
        if t_i < t_j for all neighbors j of particle i then
            IndividualStep(i)
            t_i = δt_i + 1
            compute δt_i for next iteration
            q.push(i)
        else
            wl.push(i)
        end
        cnt = cnt + 1
        if cnt == INTERVAL then
            insert all particles k in wl into queue
            empty wl
    end
```

the overhead of reconstructing queues and postponing too many particles that violate the strict particle processing order.

Waiting list handling Detemining the interval to reinsert the postponed particles back into the queue is of vital significance. An interval too small would lead to postponing these violating particles multiple times while a restoration too late would mean to postpone their neighbors in other queues and result in threads stuck at each other. An optimal choice of this hyperparameter relies heavily on the complexity of the scene and is hence done empirically.

5 Experiments

In this section, comparative experiments against previous methods including fixed time stepping (FTS), globally adaptive time stepping (GTS) [DG96] and individual stepping (ITS) [BWH+18] are conducted to demonstrate the boost on performance for the proposed method. Particles are rendered as spheres for visual comparison to avoid losing details in surface. The methods of [RHD+17] and [BGB11] are adopted for sphere and surface rendering respectively.

threads	scene	particles	FTS	GTS	ITS	ATS
			runtime	speedup	speedup	speedup
1	CDB	27k	396.9	3.0	4.4	7.5
1	CDB	125k	1896.7	2.5	3.6	7.3
6	CDB	27k	87.17	2.8	4.1	6.3
6	CDB	125k	375.9	2.2	3.2	6.8
6	RF	300k	926.7	1.3	3.0	7.5
6	FN	1.3M	3245.6	1.7	2.3	4.4
6	DB	10M	25973.0	1.8	-	6.3

Table 1: Performance comparison on various scenarios [RHEW17]. Runtimes are measured in seconds.

5.1 Performance Study

To focus on high performance, the largest possible step size that still maintains a stable simulation is used in all scenarios. Serial execution of all four methods are tested on two Corner-Dam-Break (CDB) scenes with 27k particles (in a cube of 30 x 30 x 30) and 125k particles (in a cube of 50 x 50 x 50) respectively. For parallel execution, three additional larger scenes are included to fully demonstrate the performance and scalability of the methods:

- A Radial-Flow (RF) scene. This scene features a radial force field that, serving as an external force, drives the fluid particles and generates a vortex flow. A rigid body is placed in the fluid to disturb the flow. The whole scenario consists of 300k fluid and 20k boundary particles.
- A Fountain (FN) scene. This scene features a linear force field that propels the fluid particles upwards to generate a fountain. Upon falling down, the particles collide with a slightly deformed cone to form a non-symmetric flow. The whole scenario consists of 1.3M fluid and 16k boundary particles.
- A Dam-Break (DB) scene. This scene features three collision cylinders that induces strong impact for the fluid particles. 10M particles constitute the whole scenario.

Tab. 1 exhibits the runtime and speedup factor of all four stepping methods under different scenerios. Judging from the outcome of the experiments, ATS outperforms all the other methods using either serial or parallel execution while maintaining a stable and realistic silmulation (see Fig. 7). It can be observed that the speedup factor increases as expected when the scale of the scene is enlarged. This is mainly due to the fact that the additional workload induced by handling the waiting list tend to be less influential compared to the overall computational cost when larger amount of particles are included. There is however one thing worth noting: Despite ATS still yields the best performance in the Fountain scenario, the speedup factor turns out to be less impressive than other scenes. The paper attributes the decline in performance to a lack of interation in the

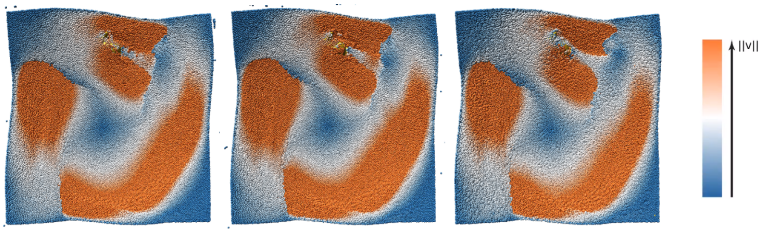

Figure 7: Visual comparison on the magnitude of particle velocities in Radial-Flow scene[RHEW17]. From left to right are the velocity fields generated by GTS, ITS and ATS respectively. Only negligible difference could be identified in the flow pattern behind the collision object.

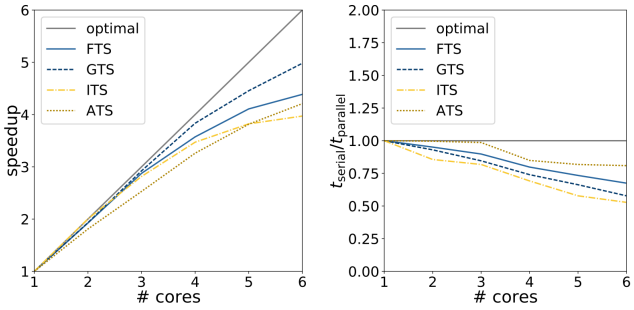

Figure 8: Visual comparison on the scalability of different time stepping models [RHEW17]. The strong scaling performance of ATS seems to be less than satisfactory thanks to its extra effort spent on queue and waiting list handling. Nevertheless, in weak scaling test it significantly outperforms all other approaches.

scene: As the fountain scenario contains a large amount of almost static particles, the speedup gained with individual time steps becomes less decisive.

5.2 Scaling Study

Another important aspect for the analysis of different time stepping models lies in their scalability. That is, how does per-thread performance change as increasingly more processing power becomes available for the solver during parallel execution? The paper conducts both strong scaling and weak scaling experiments in a comparative fashion to investigate the scalability of ATS compared to other methods. The strong scaling test measures the overall computation time to obtain the relative speedup when using up to six cores for the same task. The set of experiments are carried out on the Corner-Dam-Break scene with 125k particles. The left of Fig. 8 shows the scaling curves obtained for various approaches. ATS scales slightly worse than other methods since inevitable extra work for parallelization is employed. Nevertheless, the difference get smaller as an increasing number of cores are put into use. The paper attributes this phenomenon to

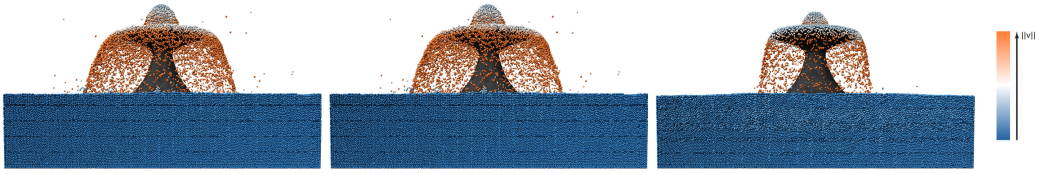

Figure 9: Measurements of the magnitude of particle velocities in Fountain scene [RHEW17]. From left to right are the velocity fields generated by GTS, ITS and ATS respectively. The waterfalls generated by various time stepping models behaves considerably different in terms of overall shape. While in ITS the waterfall leans heavily to the left to form an obviously asymmetric outlook, in ATS the velocity damping causes the particles to travel less distances after they leave the verge of the cone thus forming a steeper waterfall.

the fact that all the other methods employ simple loop parallelization: They introduce a large number of synchronization barriers into the simulation. As the number of cores grow, the solver suffers from waiting threads at such barriers and fails to maintain the relative speedup. On the other hand, weak scaling performance is worth investigating for large-scale scenarios. To this end, the paper includes a fixed number of 10k particles per core in a Corner-Dam-Break scene and examines how parallelism influences the overall simulation time as increasingly more cores are introduced. As shown in Fig. 8 (right), ATS outperforms all other methods under this test setting. As discussed above, such an outcome could be expected since the overhead introduced by queue handling in ATS becomes negligible compared to neighborhood search and the particle attribute computations when handling larger number of particles.

5.3 Visual Comparison

The paper make some visual comparisons on the generated animations to qualitatively analyze the behavior of fluid particles. In particular, it is noticed that ATS dampens the fluid velocities to a small extent where interaction occurs. In the Radial-Flow scene such damping effect may be observed in the disturbed flow regions around the collision object 7. But since an external force field constantly exerts action on the particles, such minor difference fade out soon after the collision. However, in scenarios where no external driving forces occur, such damping leads to visually obvious discrepancies in particle behavior. In the Fountain scenario 9, due to dampened velocities, particles that fall down after collision with the cone form an almost symmetric waterfall that is considerably smaller in size. Additionally, the particles that are propelled upwards at the top of the cone reach only lower heights.

To analyze this effect, the paper computes over time the mean velocity magnitudes of particles in the collision area for both scenes. In the Radial-Flow scenario only particles near the bunny are investigated and in the Fountain test only particles above the pond. The obtained curves are plotted in Fig. 10. Overall, the difference between velocity damping across various time stepping schemes coincide with previous visual

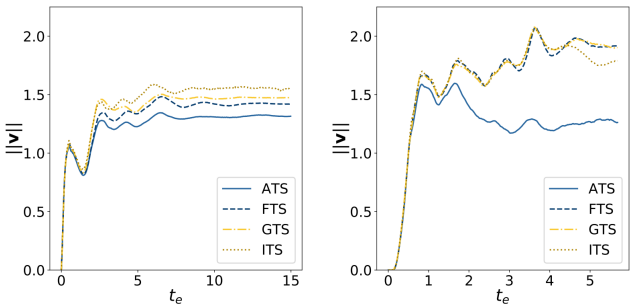

Figure 10: Quantitative comparison on the mean velocities of interacting particles in the Radial-Flow scene (left) and Fountain scene (right) [RHEW17]. In both scenes, ATS demonstrate the most obvious damping effect among all tested methods.

inspection. It is worth noting in the Fountain test however, that in ATS the particles only experienced decelerations after initial impact with the cone, implying that proposed asynchronous stepping method does not seem to handle the interactions between fluid particles very well. In this regard, perhaps further improvements on the time integration scheme of the method could be explored.

Apart from the aforementioned flaws, ATS still serves as an equally reliable method to generate visually appealing fluid animation. Especially in regions where heavy interactions occur, the simulation remains stable and the particles travel smoothly even when the largest possible λ_v and λ_F constants are used.

6 Conclusion

In this paper, the paper presents a novel asynchronous time stepping method for non-iterative SPH integration. Each particle is at each time given a dedicated step size and no global synchronization barriers are introduced to avoid time wasted by waiting threads in previous works. To guarantee local consistency and support attribute computation, neighborhood reconstruction is done each time a particle is processed. A strict chronological processing order of the particles in the form of a priority queue is introduced to enhance stability. A multi-queue-based scheme is smartly devised for parallel execution of this fully asynchronous time integration method. In terms of efficiency, the proposed algorithm is capable of achieving a speedup factor of up to 7.5 and outperforms other tested methods under various scenarios using either serial or parallel execution. In terms of scalability and visual correctness, the respective tests yield some drawbacks of the integration scheme that could be further exploited:

- With large proportions of barely moving fluid particles the algorithm's performance boost is undermined. Here the promising work of [GB14] could be adopted so that the solver handles barely active particles as static ones.

- As pointed out by [BET+10], the acceleration term in Eq. (8) dominates the speed term in non-iterative SPH simulation. A natural idea would be to devise a scheme such that the constants λ_v and λ_F are automatically adapted for the particles according to their neighborhood information.
- Compared to other tested models, asynchronous time stepping leads to higher velocity damping upon interaction with collision objects. Since the paper only implemented a simple boundary handling scheme, in which the particles at domain boundaries are clipped in position, reversed in velocity and applied with a user-defined friction, such damping could be alleviated if a more complex boundary handling model is adopted.

References

[AIA+12] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible sph. *ACM Transactions on Graphics (TOG)*, 31(4):62, 2012.

[BET+10] J Bender, K Erleben, M Teschner, et al. Boundary handling and adaptive time-stepping for pcisph. In *Workshop on virtual reality interaction and physical simulation VRIPHYS*, 2010.

[BGB11] Haimasree Bhatacharya, Yue Gao, and Adam Bargteil. A level-set method for skinning animated particle data. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 17–24. ACM, 2011.

[BWH+18] Xiaojuan Ban, Xiaokun Wang, Liangliang He, Yalan Zhang, and Lipeng Wang. Adaptively stepped sph for fluid animation based on asynchronous time integration. *Neural Computing and Applications*, 29(1):33–42, 2018.

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation'96*, pages 61–76. Springer, 1996.

[GB14] Prashant Goswami and Christopher Batty. Regional time stepping for sph. In *Eurographics 2014*, pages 45–48. Eurographics Association, 2014.

[HRWE15] Markus Huber, Stefan Reinhardt, Daniel Weiskopf, and Bernhard Eberhardt. Evaluation of surface tension models for sph-based fluid animations using a benchmark test. In *VRIPHYS*, pages 41–50, 2015.

[ICS+13] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible sph. *IEEE transactions on visualization and computer graphics*, 20(3):426–435, 2013.

[IOS+14] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. Sph fluids in computer graphics. 2014.

[Mon92] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30(1):543–574, 1992.

[RHD+17] Stefan Reinhardt, Markus Huber, Otilia Dumitrescu, Michael Krone, Bernhard Eberhardt, and Daniel Weiskopf. Visual debugging of sph simulations. In *2017 21st International Conference Information Visualisation (IV)*, pages 117–126. IEEE, 2017.

[RHEW17] Stefan Reinhardt, Markus Huber, Bernhard Eberhardt, and Daniel Weiskopf. Fully asynchronous sph simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 2. ACM, 2017.