Fully Asynchronous SPH Simulation

Stefan Reinhardt
Stuttgart Media University
reinhardts@hdm-stuttgart.de

Markus Huber
VISUS, University of Stuttgart
markus.huber@visus.uni-stuttgart.de

Bernhard Eberhardt
Stuttgart Media University
eberhardt@hdm-stuttgart.de

Daniel Weiskopf
VISUS, University of Stuttgart
weiskopf@visus.uni-stuttgart.de

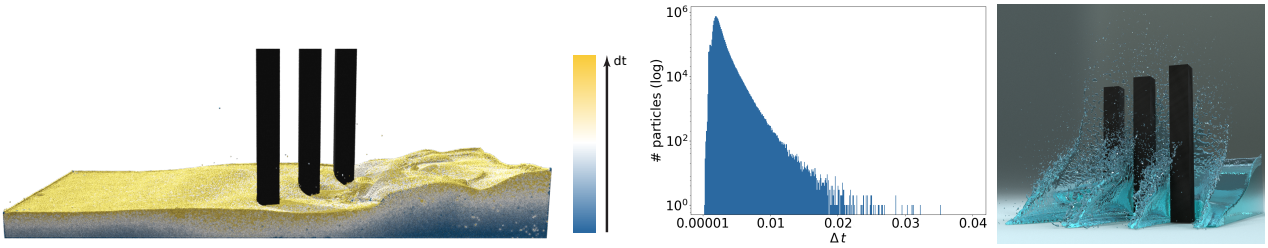


Figure 1: In SPH-based fluid simulation, the maximum possible time step size varies across particles as shown in the histogram (center). While previous methods use the global minimum to advance the particles, we allow a dedicated time step for each particle through asynchronous time integration. In the left figure, the used time step is color-coded for a 10 *M* particle simulation. For this large-scale scenario, we achieve speedups of a factor of 6.3 compared to a fixed time step. A production rendering is shown in the right figure.

ABSTRACT

We present a novel method for fully asynchronous time integration of particle-based fluids using smoothed particle hydrodynamics (SPH). With our approach, we allow a dedicated time step for each particle. Therefore, we are able to increase the efficiency of simulations. Previous approaches of locally adaptive time steps have shown promising results in the form of increased time steps, however, they need to synchronize time steps in recurring intervals, which involves either interpolation operations or matching time steps. With our method, time steps are asynchronous through the whole simulation and no global time barriers are needed. In addition, we present an efficient method for parallelization of our novel asynchronous time integration. For both serial and parallel execution, we achieve speedups of up to 7.5 compared to fixed time steps and are able to outperform previous adaptive approaches considerably.

CCS CONCEPTS

• **Computing methodologies** → **Animation**; **Physical simulation**;

KEYWORDS

Fluid simulation, smoothed particle hydrodynamics, asynchronous time integration

ACM Reference format:
Stefan Reinhardt, Markus Huber, Bernhard Eberhardt, and Daniel Weiskopf. 2017. Fully Asynchronous SPH Simulation. In *Proceedings of SCA '17, Los Angeles, CA, USA, July 28-30, 2017,* 10 pages.
DOI: 10.1145/3099564.3099571

SCA '17, Los Angeles, CA, USA

978-1-4503-5091-4/17/07…$15.00
DOI: 10.1145/3099564.3099571

1 INTRODUCTION

In recent years, smoothed particle hydrodynamics (SPH) has evolved to an established method in computer graphics for the animation of fluids. Due to its Lagrangian nature, it is suitable to simulate free surfaces and complex interactions. To meet the demand of the highest possible visual quality of animations, particle numbers keep increasing. Although the computational effort per simulation step is not excessively high, performing simulations with tens of millions of particles in complex scenarios can take hours or even days on state-of-the-art workstations.

In scenarios that induce only little error in density deviation, non-iterative SPH solvers that employ an equation of state (EOS) can provide very good performance, as discussed by Ihmsen et al. [2014b]. To ensure realistic animations, the stiffness constant in the EOS has to be chosen quite large. A large stiffness constant, however, requires small steps for time integration to guarantee stability of the simulation and leads to significant computational costs. The maximum possible time step is bound by the Courant-Friedrichs-Lewy (CFL) condition and depends on the particle radius and the maximum particle velocity. Most SPH solvers use a uniform time step for every particle that can be adapted according to the

Figure 2: A simulation with asynchronous time integration using 300 k particles with a radial force field and a bunny as a collision object to disturb the flow. We achieve a speedup of a factor of 7.5 compared to fixed time stepping.

CFL condition in each simulation step. For stability reasons, this restriction may only be necessary for a few particles. As shown in the histogram in Fig. 1, the maximum possible time steps may vary substantially. As they cover up to three magnitudes, much unnecessary computational effort is spent when using a fixed or globally adaptive time stepping method.

We present a fully asynchronous time integration scheme for non-iterative EOS solvers that allows for dedicated time steps for each particle. By handling the time step restriction for each particle individually, computational resources are focused on important regions and the overall computation time can be reduced considerably while ensuring stability. Fig. 2 shows a typical example of asynchronous time integration, where we achieve a speedup of a factor of 7.5 compared to fixed time stepping. In comparison to previous approaches for locally adaptive time integration, we avoid synchronizing particle states at time barriers [Desbrun and Gascuel 1996][Desbrun and Cani 1999][Ban et al. 2016] at any time during simulation, nor do we divide the fluid domain into regions [Goswami and Batty 2014].

The main technical contributions of this paper are:

- We present a new, fully asynchronous time integration model for SPH-based fluid animation. In contrast to other methods for local time adaptivity, our method does not need global synchronization barriers. Throughout the simulation, each particle has its dedicated time step depending on the state of the simulation and particles are processed using a priority queue.
- While SPH simulations with a uniform time step for every particle can easily be parallelized on shared memory architectures, parallelism with asynchronous integration is not straightforward. We introduce a parallelization technique for asynchronous time integration using multiple queues.

2 RELATED WORK

There are different approaches to simulate SPH-based fluids used in computer graphics applications. A classification and overview of algorithms are presented by Ihmsen et al. [2014b]. To increase the computational efficiency of simulations, especially for large particle numbers, adaptive models provide a suitable solution. An overview of **adaptive** models in computer graphics in general can be found in the survey of Manteaux et al. [2016].

To employ **spatial adaptivity** in SPH, different concepts have been proposed. Desbrun and Cani [1999] use a basic splitting and merging concept to increase the detail only in areas of interest. Adams et al. [2007] introduce a condition based on geometric local feature sizes for splitting and merging, and vary particle sampling by weighting according to visual importance. With the approach of Ando et al. [2012], particles inside the fluid are collapsed and a high-resolution simulation is performed on the surface only. To preserve thin features, they base the resampling criterion on the anisotropy of the particles neighborhood. Orthmann and Kolb [2012] use diffusive flux as a resampling criterion. Another approach to spatial adaptivity employs coupled simulations at different scales as proposed by Solenthaler and Gross [2011]. They perform a fine-scale simulation on predefined regions that can dynamically change. As this approach does not guarantee mass conservation, Horvath and Solenthaler [2013] propose an emission scheme that tracks the total mass and iteratively compensates volume changes.

In this work, we focus on **temporal adaptivity**. A straightforward approach is to determine a globally adaptive time step, using the CFL condition as proposed by Desbrun and Gascuel [1996]. This is also used for predictive-corrective incompressible SPH (PCISPH) with additional shock handling [Ihmsen et al. 2010]. Goswami and Batty [2014] introduce a regional time integration model. Regions with individual time steps are defined and particles in this region are advanced accordingly. They define a base time step Δt_b and restrict the used time steps to a multiple of Δt_b, hence, introducing equidistant buckets. Again, Desbrun and Gascuel [1996] propose using an individual time step per particle. Based on a reformulation of the CFL condition, the simulation time step is subdivided by powers of two. Forces are evaluated only if required by the stability criterion, and positions are integrated with the smallest time step. Ban et al. [2016] follow this idea and compute the time step for each particle, hence employing individual time stepping. If the individual time step of a particle is smaller than a global time step, it is set active and evaluated in the SPH loop. Otherwise, the particle is marked as inactive and is linearly interpolated to the next global time step. As in globally adaptive time stepping, the global time step is determined by the minimal individual time step. Particles are synchronized at every export sample as presented by Desbrun and Gascuel [1996]. In our approach, we use an asynchronous time integration method that does not need global synchronization. In Sec. 5, we compare our method to globally adaptive and individual time stepping.

For elastodynamics, asynchronous variational time integration (AVI) allows substantial speedups without reducing accuracy [Lew et al. 2004]. In computer graphics, AVIs have been successfully applied to cloth simulations including asynchronous collision handling [Thomaszewski et al. 2008][Harmon et al. 2009]. Schroeder et al. [2011] propose a hybrid implicit and semi-implicit asynchronous time integration method to simulate deformable objects. Asynchronous integration is also used with fully implicit time integration [Zhao et al. 2016]. They all employ AVI on finite element simulation models with predetermined neighborhood information.

Algorithm 1 Single SPH simulation step with splitting

```
function DO_EOS_STEP()
    for each particle i do
        find neighbors j
    for each particle i do
        compute advection force F_i^* = F_i^viscosity + F_i^ext
        compute advection velocity v_i^* using F_i^*
    for each particle i do
        compute advection density ρ_i^*
        compute pressure p_i
    for each particle i do
        compute pressure force F_i^p
    for each particle i do
        compute new particle velocity v_i(t + Δt)
        compute new particle position x_i(t + Δt)
```

We present an asynchronous integration method for Lagrangian flow with varying neighborhood.

Speeding up SPH simulations can also be achieved by exploiting parallelism. On shared memory systems, SPH simulation models using global synchronizations can be parallelized using simple loop parallelization. Efficient data structures for parallel neighborhood query and processing are discussed in the survey of Ihmsen et al. [2011]. Thaler et al. [2014] present an architecture to compute implicit incompressible SPH simulations in parallel on distributed memory systems. Kale and Lew [2007] propose a scheme for the parallel execution of AVI for elastodynamic objects. We build upon their idea of multiple parallel queues to parallelize asynchronous time integration for SPH. For load balancing, they decompose the domain by solving a graph partitioning problem using weighted edges for the predetermined neighbors. Since in Lagrangian flow no predetermined neighborhood exists, we propose an over-decomposition of the problem size.

3 SIMULATION MODEL

Fluid motion can be described by the Navier-Stokes equations and SPH is used to calculate its components. The fluid properties $A(\mathbf{x}_i)$ are evaluated at discrete particle positions $\mathbf{x}_i$ and interpolated using a kernel function W of compact support. After calculating fluid properties, the particles are moved forward in time. There exist various approaches to solve the source terms of the Navier-Stokes equations with SPH. In the following, we recapitulate the necessary concepts for our work, describe our used model, and show how we integrate particle locations over time.

3.1 SPH model

Non-iterative SPH solvers usually determine the pressure by a state equation. In our work, we use the state equation presented by Desbrun and Gascuel [1996]. Additionally, we split the computation of advection forces $\mathbf{F}^*$ and pressure forces $\mathbf{F}^p$ as described by Ihmsen et al. [2014b]. Using the splitting concept, advection density ρ^* is calculated based on advection forces, which is then used to calculate pressure. Hence, advection forces are implicitly considered in the pressure force calculation. Algorithm 1 describes the outline of one simulation step using the splitting concept.

Algorithm 2 The asynchronous simulation step of one particle

```
function DO_STEP(i)
    determine possible time step Δt_i
    reconstruct neighbor attributes A_j(t_i)
    compute ρ_i
    compute F_i^*
    compute v_i^* using F_i^*
    compute ρ_i^*
    compute p_i = k(ρ_i^* − ρ_0) and pressure forces F_i^p
    integrate particle i over time using Δt_i
```

To compute the spatial derivatives of the fluid properties, we use the approximations proposed by Monhagan [1992]. Thus, we compute the viscosity forces acting on particle i via

$$\mathbf{F}_i^{viscosity} = 2m_i\nu \sum_j \frac{m_j}{\rho_j}\mathbf{v}_{ij}\frac{\mathbf{x}_{ij}\cdot\nabla W_{ij}}{\mathbf{x}_{ij}\cdot\mathbf{x}_{ij}+0.01h^2}, \tag{1}$$

where m_i is the mass of particle i, ρ_j is the density of particle j, and $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$ and $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ are relative velocities and positions, respectively. ν is the viscosity coefficient and h is the smoothing kernel length. ∇W_{ij} is the gradient of the smoothing kernel $W_{ij} = W(\|\mathbf{x}_{ij}\|, h)$ and describes the influence of particle j on the attribute calculation of particle i. After computing advection forces $\mathbf{F}_i^*$, the intermediate velocities $\mathbf{v}_i^*$ are calculated and used to determine the advection density ρ_i^* via

$$\rho_i^* = \sum_j m_j W_{ij} + \Delta t_i \sum_j (\mathbf{v}_i^* - \mathbf{v}_j^*)\cdot\nabla W_{ij}. \tag{2}$$

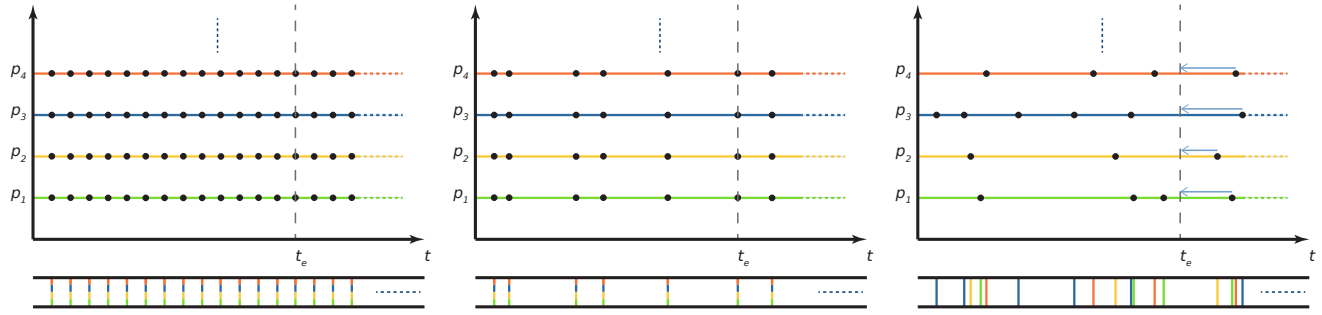
In this equation, the first summation considers the density ρ_i and the second one its temporal derivation $\frac{d\rho_i}{dt}$. After advecting the particles, we compute the pressure acting on particle i via $p_i = k(\rho_i^* - \rho_0)$, where ρ_0 is the system's reference density and k is the stiffness constant that scales the pressure. Negative pressures are clamped to zero, which is common when using an EOS [Ihmsen et al. 2014a]. The respective pressure forces are computed using

$$\mathbf{F}_i^p = -m_i \sum_j m_j \left(\frac{p_i}{(\rho_i^*)^2} + \frac{p_j}{(\rho_j^*)^2}\right)\nabla W_{ij}. \tag{3}$$

The stiffness constant k scales the pressure. High values reduce the compressibility of the fluid and therefore, increase pressure forces. As mentioned in Sec. 1, high values for k require small time steps.

We employ interaction with static objects using the boundary handling model of Akinci et al. [2012]. At domain boundaries, we clip the particle positions, reflect their velocities, and add user-defined boundary friction. A typical scenario is shown in Fig. 2, where a radial flow is present and a collision object is placed into the scene to disturb the flow. To produce the radial flow we include force fields into the simulation. They are added as external forces and contribute to the advection force $\mathbf{F}_i^*$.

Surface tension has a high influence on the visual appearance of the fluid and is important to be included into the simulation [Akinci et al. 2013; Becker and Teschner 2007; He et al. 2014; Huber et al. 2015]. In our experiments, we incorporate surface tension using a



(a) FTS (b) GTS (c) ATS

Figure 3: Illustration of different time stepping methods: (a) fixed time stepping (FTS), (b) globally adaptive time stepping (GTS), and (c) our asynchronous time stepping (ATS). With FTS and GTS, particles p_i are updated synchronously, based on the minimum time step. We use varying time steps for time evolution of different particles, which leads to a strict particle processing order using a priority queue ((c) bottom). No global synchronization is needed and we only employ virtual barriers to export simulation states (here, at time t_e). By enlarging the time step size for each particle, we reduce the overall amount of operations.

modification of the inter-particle interaction force model [Becker and Teschner 2007], as proposed by Huber et al. [2015].

3.2 Time integration

To integrate particle attributes over time, different numerical integration schemes such as the explicit Euler, semi-implicit Euler, or a variant of the Verlet method can be used. They are all suitable for the time integration of our system. The semi-implicit Euler, also known as symplectic Euler, is one of the most commonly used time integration schemes [Ihmsen et al. 2014b] and is also employed here. Similarly to Goswami and Batty [2014], we add a modification: the computation of $\mathbf{x}_i(t + \Delta t)$ is a first-order Taylor approximation and we include the second-order term to compute the new particle positions. The integration scheme then reads as

$$\begin{aligned} \mathbf{v}_i(t + \Delta t) &= \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i(t)}{m_i}, \\ \mathbf{x}_i(t + \Delta t) &= \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t) + \frac{\Delta t^2}{2} \frac{\mathbf{F}_i(t)}{m_i}, \end{aligned} \tag{4}$$

where Δt denotes the size of the time integration step. In our experiments, this modification enhances the stability of the time integration step and, hence, allows for larger time steps. However, this integration scheme is no longer symplectic as it introduces numerical damping. The possible time step size Δt itself is governed by the CFL condition: $\Delta t \leq \lambda \frac{2r_i}{\|\mathbf{v}^{max}\|}$, with r_i being the particle radius, λ a constant between zero and one, and $\mathbf{v}^{max}$ the maximum fluid velocity. For stability, acting forces should additionally be considered to compute the time step for particle i [Ihmsen et al. 2010; Monaghan 1992]. The possible time step for particle i can then be determined via

$$\Delta t_i = \min\left(\lambda_{\mathbf{v}} \frac{2r_i}{\|\mathbf{v}_i\|}, \lambda_{\mathbf{F}} \sqrt{\frac{2r_i}{\mathbf{F}_i^{total}/m_i}}\right), \tag{5}$$

where $\lambda_{\mathbf{v}}$ restricts the velocity and $\lambda_{\mathbf{F}}$ the force of particle i. In previous approaches, the system-wide time step is then calculated as the minimum of individual possible time steps $\Delta t = \min_i(\Delta t_i)$.

4 ASYNCHRONOUS TIME INTEGRATION

In this section, we present our novel approach for the asynchronous simulation of SPH-based fluids. We propose a time integration model in which each particle i can be integrated with a dedicated step Δt_i determined by Eq. 5. Using the common approach for particle processing as shown in Algorithm 1, the particles' attribute computations need to be synchronized several times during one simulation step. This introduces a large amount of synchronization barriers as illustrated by the combined particle updates in Figs. 3 (a) and (b). Each particle has to be processed five times per update to compute its new state. With our approach, we are able to compute the new state of a particle at once, as outlined in Algorithm 2. Furthermore, there is no need for a global synchronization of the simulation and we do not have a global simulation state. We only add a virtual barrier to export specific states of the simulation for visualization. Our asynchronous particle handling follows the concept presented by Lew et al. [2004] and enforces a strict particle processing order using a priority queue. Fig. 3 (c) illustrates our time integration model and the particle processing order.

4.1 Asynchronous particle handling

SPH-based fluids consist of individual particles that are locally influenced by their surrounding neighbors. This means that the problem is of compact support and the particles can be handled individually, as long as the needed neighborhood information remains consistent. As each particle has its own time stamp t_i, it usually differs from the time stamps t_j of its neighbors, which is illustrated in Fig. 4. To compute the fluid properties at particle i, we need to transform the neighborhood into a consistent state. This is done by reconstructing the attribute values $A_j(t_i)$ of the neighboring particles j at time t_i

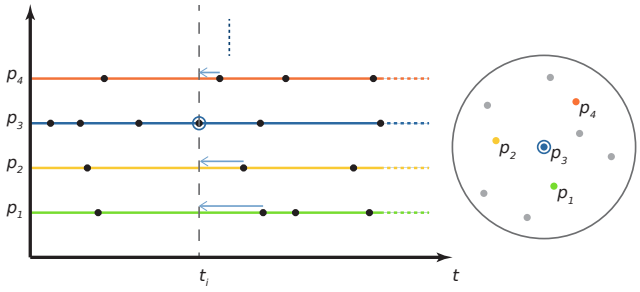


Figure 4: Neighborhood reconstruction in asynchronous time integration for the example of particle p_3: on the right, neighbors contributing to the attribute computation are shown and on left, the individual time steps. Attributes $A_j(t_j)$ are traced back in time to t_i to compute $A_i(t_i)$.

as illustrated in Fig. 4. The attributes of neighboring particles are traced back in time using their derivatives:

$$A_j(t_i) = A_j(t_j) + (t_i - t_j)\frac{dA_j(t_j)}{dt}. \tag{6}$$

We need to compute the following attributes from neighboring particles to obtain the fluid properties at particle i: mass $m_j(t_i)$, density $\rho_j(t_i)$, pressure $p_j(t_i)$, position $\mathbf{x}_j(t_i)$, and velocity $\mathbf{v}_j(t_i)$. $\mathbf{x}_j(t_i)$ and $\mathbf{v}_j(t_i)$ are reconstructed using the already computed forces and the scheme described in Eq. 4. Since we use the splitting concept, we have already computed the derivative $\frac{d\rho_j}{dt}$ to determine the advection density and are able to reconstruct $\rho_j(t_i)$ without any extra cost. Pressure $p_j(t_i)$ is computed on the fly based on the EOS. Mass m_j is constant and does not have to be computed.

After synchronizing the neighborhood, we are able to compute the attributes of particle i including time integration, while processing it only once. An outline of the asynchronous step is given in Algorithm 2.

To handle all attributes at once and to ensure that no important information is missed in the simulation (such as shocks), we add one important constraint to our time integration model: we employ the strict particle processing order $t_i \leq t_j$ for all neighbors j. This also ensures that we trace neighbors only back in time, which further enhances the stability of our simulation system.

To identify the particle that has to be processed next, we employ a priority queue, where particle indices are stored and sorted by their current simulation time. Fig. 3 (c) illustrates how we move the particles forward in time and the particle queue is depicted at the bottom of the figure. We dequeue the frontmost particle and compute its attributes as described in Algorithm 2. Then, the particle is pushed back into the queue with its new time stamp and to be sorted within the queue. The top row of Fig. 3 shows the time line of selected particles. Here, we can see that our model does not introduce any global barriers into the simulation. We only add a virtual, non-invasive barrier to export the current simulation state at t_e: once the last particle has reached t_e, we are able to store the current simulation state. For export, the particle attributes are traced back to t_e using the same reconstruction scheme as used for the neighborhood synchronization (Eq. 6).

Algorithm 3 Particle handling using a priority queue (operations for parallel execution are highlighted in blue color)

```
function RUN_QUEUE(queue,t_e)
    Initialize waiting list wl
    Initialize step counter ctr
    while t_queue.top() < t_e do
        i = queue.pop()
        find neighbors j
        if t_i > t_j then                    ▷ t_i ≤ t_j is not guaranteed
                                               for multiple queues
            wl.push(i)
        else
            DO_STEP(i)
            t_i+ = Δt_i
            compute Δt_i^new (Eq. 5)
        queue.push(i)
        ctr = ctr + 1
        if ctr%⌊0.025 · queue.size()⌋ == 0 then
            insert all particles i of waiting list wl into queue
```

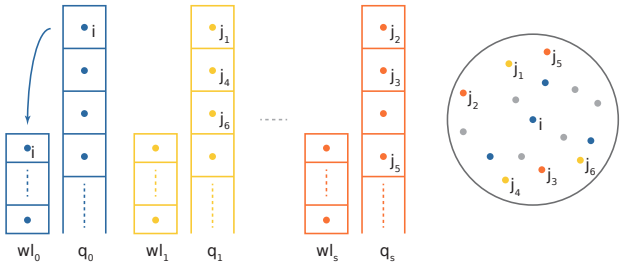


Figure 5: Our parallelization concept: when using multiple queues, our strict particle processing order may be violated. We enforce it by introducing a waiting list per queue. If the condition $t_i \leq t_j$ is violated by any neighboring particle, the current particle i is pushed into the waiting list.

Algorithm 3 presents the queue concept. The queue is processed until all particles reach the next t_e and the simulation state is exported. After exporting, we repeat this step until the end of the simulation. Algorithm 3 also includes our parallelization concept, which we will discuss in the following section. The additional operations needed for parallel processing are highlighted in blue color.

4.2 Parallelization

As modern hardware supports multi processing, we show how our model can be adapted to being executed in parallel. The parallelization of a usual SPH step (Algorithm 1) can be achieved easily by parallel execution of the loops. Using our model, this is not possible since we do not have a predetermined loop count. We follow the concept of Kale and Lew [2007] and parallelize the simulation by using multiple queues q_a. If b is the number of used threads, we spawn at least b queues. Spawning b queues is sufficient for particle counts up to around $1\,M$. For large scale scenarios, such as the

Table 1: Runtime comparison: we conduct performance tests on five different scenarios. A $27\,k$ and $125\,k$ Corner Dam Break, the radial flow (Fig. 7), the fountain scenario (Fig. 8), and a large-scale dam break (Fig. 1). Runtime measurements are given in seconds and present computation times for $1\,s$ simulation using fixed time stepping (FTS), globally adaptive time stepping (GTS), individual time stepping (ITS), and fully asynchronous time stepping (ATS). For FTS, we set $dt = 0.25\,ms$ for all scenarios.

	Scene	# particles	FTS	GTS		ITS		ATS (ours)	
			runtime	runtime	speedup	runtime	speedup	runtime	speedup
1 thread	Corner Dam Break	$27\,k$	396.9	130.4	3.0	89.7	4.4	52.7	7.5
	Corner Dam Break	$125\,k$	1896.7	768.1	2.5	532.13	3.6	260.4	7.3
6 threads	Corner Dam Break	$27\,k$	87.17	30.8	2.8	21.12	4.1	13.9	6.3
	Corner Dam Break	$125\,k$	375.9	170.8	2.2	117.6	3.2	55.6	6.8
	Radial flow	$300\,k$	926.7	723.9	1.3	304.2	3.0	123.3	7.5
	Fountain	$1.3\,M$	3245.6	1954.0	1.7	1427.2	2.3	740.6	4.4
	Dam Break	$10\,M$	25973.0	14560.0	1.8	-	-	4106.9	6.3

$10\,M$ corner dam break depicted in Fig. 1, the solver suffers from waiting threads and we need to consider load balancing. Kale and Lew [2007] calculate the computational cost per element and additionally use neighbor relations to perform domain decomposition to distribute computational effort uniformly. This is not possible with our approach, as we do not have a predetermined neighborhood.

We address this issue by using over-decomposition of the problem size and spawning at least three queues per thread. The concept of multiple queues is depicted in Fig. 5. The individual queues are handled in parallel. Since the different priority queues do not communicate, our enforced particle processing order ($t_i \leq t_j$) may be violated. If a neighbor j of the current particle i is held in a different queue, we cannot guarantee the correct processing order. To resolve this issue we introduce a waiting list wl_{q_a} per queue. We remove the particle i from the queue and add it to the waiting list, if any of its neighbors has not reached the necessary simulation time t_i as shown in Fig. 5. We empty the list and reinsert the particles back into the queue in regular intervals and, thus, are able to reinforce our processing order. The parallel asynchronous process of one queue between two export steps is shown in Algorithm 3.

4.3 Implementation details

At first glance, organizing particles in a queue seems to be expensive, but storing only particle indices, and using a heap to organize the queue, grants efficient access to the next processed particle. The attributes of the particles are stored in contiguous arrays.

For the neighborhood search, we use a uniform grid storing the cells in a one-dimensional array using Z-index ordering [Ihmsen et al. 2011]. Since a global update of the neighbor cells after processing each particle would be expensive, we only process a global update at each virtual export barrier. To avoid missing a neighbor contributing to the attribute computation, we check if the particle has changed the cell after processing it. If this is true, the particle is added into the new cell without removing it from its previous cell. Combined with our strict particle processing order, we can ensure that no neighboring particles are missed.

The introduced waiting list adds a slight amount of extra work, especially, if many of the neighboring particles are stored in other queues. As we employ Z-index ordering of the particle arrays, we can simply split the queues by particle index and, therefore, achieve spatial decomposition. Since particles move during the simulation, this separation will be reduced. Therefore, we reorder the particles at specific time stamps. Then, we reinitialize the queues and increase the particle separation across the different queues again. We only perform reinitialization after several virtual export barriers. In our experiments, a reinitialization of the queues q_a is done after t_e reaches a multiple of $0.5\,s$, which provides a good trade-off between the overhead of reinitializing the queues and postponing particles that violate the strict particle processing order. The global reconstruction is performed without particle synchronization and enhances the performance of our time integration method.

Defining the time of reinserting the postponed particles into the queue is crucial for performance optimization. Reinserting them too early could lead to postponing these particles multiple times. In contrast, reinserting them too late could lead to unnecessary postpone actions in other queues. Unfortunately, there is no predefined value for this issue as the optimal size of waiting list wl_{q_a} depends on the scenario and needs some testing. In our experiments, setting the size to 2.5 % of the queue size has proved to work well in almost every case.

To initialize the simulation, we perform one global step and determine the individual Δt_i. For time step restrictions, Ihmsen et al. [2010] propose using $\lambda_{\mathbf{v}} = 0.4$ and $\lambda_{\mathbf{F}} = 0.25$, as also employed by Monhagan [1992]. Since we use an EOS, we have to deal with stiff forces and need to reduce these constants. Ban et al. [2016] use $\lambda_{\mathbf{v}} = 0.1$ and $\lambda_{\mathbf{F}} = 0.025$. Employing the splitting concept, we were able to enlarge the constants to $\lambda_{\mathbf{v}} = 0.25$ and $\lambda_{\mathbf{F}} = 0.05$. Similar to Goswami and Batty [2014], we clip the step sizes to be a multiple of $0.5\,ms$, hence, introducing equidistant buckets of time steps. Additionally, we use $\Delta t_i = \min_j(\Delta t_j)$ to improve stability at high impact velocities.

5 RESULTS

In this section, we provide results obtained with our asynchronous time stepping (ATS) and compare them to previous methods: fixed time stepping (FTS), globally adaptive time stepping (GTS) [Desbrun and Gascuel 1996], and individual stepping (ITS) [Ban et al. 2016]. First, we investigate performance, followed by a visual comparison.

All results are performed on a workstation with an Intel Xeon E5-1650 v3 hexacore CPU at 3.5 GHz and 64 GB DDR4 memory. For

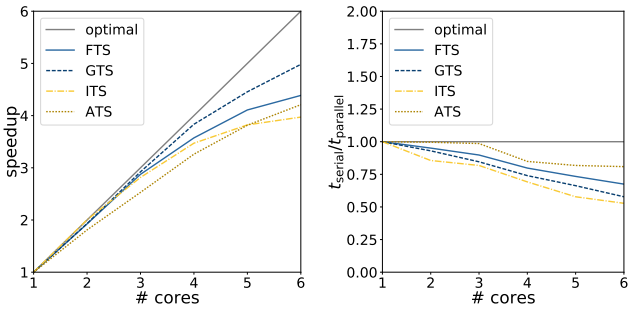


Figure 6: Scaling test: strong scaling (left) and weak scaling (right). In strong scaling, our model scales slightly worse compared to other methods using only a few cores, but as the number of cores increases, it outperforms ITS. On weak scaling, all other tested methods are outperformed.

parallelization, we use OpenMP to employ loop parallelization. We implemented all tested approaches in our simulation system and used the same code base for all simulations. For visual comparison, we render the particles as spheres to avoid losing details in surface reconstruction. The sphere rendering is performed with a custom plugin for MegaMol [Grottel et al. 2015] tailored for SPH simulations [Reinhardt et al. 2017] that allows for interactive feedback for up to several million particles and also incorporates analysis tools. For surface rendering, we apply the level-set technique by Bhattacharya et al. [2015] to extract the fluid surface.

If not stated otherwise, we use the smoothing kernel presented by Müller et al. [2003]. The viscosity parameter is set to $\nu = 0.01$, and the surface tension coefficient to $\varphi = 0.02$.

5.1 Performance comparison

We conduct our performance study separately for serial and parallel execution. Since our method processes particles differently than previous methods, we first investigate runtime using serial code. Then, performance and scalability using the parallelization presented in Sec. 4.2 is analyzed.

We use four different simulations with the number of particles varying from $27\,k$ up to $10\,M$ and different characteristics. The first example is a corner dam break (CDB) with $27\,k$ particles arranged in a cube of $30 \times 30 \times 30$ particles. This scenario is then scaled up to $125\,k$ ($50 \times 50 \times 50$) particles. Visualizations of this scenario can be found in the supplemental material. In the parallel performance comparison, we additionally include three larger scenarios. The first example consists of $300\,k$ fluid and $20\,k$ boundary particles as shown in Fig. 2. In this test case, a radial force field is added to external forces to generate an almost homogeneous flow disturbed by a collision object. The second scenario consists of $1.3\,M$ fluid and $16\,k$ boundary particles as shown in Fig. 8. The collision object is a slightly deformed cone. We employ a linear force field to produce the fountain. Due to the deformed shape of the cone the splash on the top starts to move around and a non-symmetric flow is formed. In this scenario, most of the particles are barely moving and only few are involved in interaction. A large-scale scenario with $10\,M$ particles (see Fig. 1) is included in the performance comparison to prove the scalability of our method. As we focus on high performance, we use the largest possible time step size that still maintains a stable simulation in all scenarios.

In Table 1, overall runtime and speedup of our method compared to FTS, GTS, and ITS are given for all test scenes. Using serial execution, our method is approximately 7.5 times faster than FTS, around three times faster than GTS, and up to twice as fast as ITS. Parallelization introduces computational overhead and the speedup of all tested methods is slightly reduced. In our model, additional work is added to the solver when introducing the waiting lists wl_{q_a}. Even though the speedup decreases, the difference to GTS and ITS is only minor and our method outperforms all other tested methods.

If the scenes are getting larger, the speedup increases because the additional workload becomes smaller compared to the overall computational cost. In the radial flow scenario (Fig. 7), the performance gain of our method compared to GTS and ITS is apparent. For the fountain scene (Fig. 8), our method is still faster than the compared methods, however, the speedup is less in this situation. While still being almost twice as fast as ITS, our method maintains only an overall speedup of a factor of 4.4. We notice that our model shows best performance, if there is much interaction in the scene. As the fountain scenario consists of a large amount of barely moving particles, the speedup gained with our model slightly decreases.

We also test our method on a $10\,M$ dam break with collision objects as shown in Fig. 1. We observe almost linear scaling of the computation time when increasing the number of particles. As shown in Table 1, we obtain a speedup of a factor of 6.3 compared to FTS, and outperform GTS by a factor of 3.5. In absolute numbers, this means that we are able to reduce simulation time from 7.22 hours using fixed stepping to 1.14 hours with our method.

Further, we conduct a scaling study measuring strong and weak scaling of all methods. The strong scaling study is performed on the $125\,k$ CDB. We measure the overall computation time for the simulation of $3\,s$ and compute the average over three simulation runs. The number of cores is increased from one up to six and the relative speedup is computed. The left of Fig. 6 shows the comparison of the different time integration methods. As we employ extra work to the parallelization, our method scales slightly worse than the other methods. However, with an increasing number of cores, the difference gets smaller. Using six cores, we are able to get a higher speedup compared to ITS. FTS, GTS, and ITS first scale well with few cores, but cannot maintain the relative speedup if the number of cores increases. These methods are based on simple loop parallelization. They introduce quite a number of barriers into the simulation, as they have to synchronize multiple times per step. Therefore, scaling declines with an increasing number of cores. Our method only enforces a virtual barrier at the export time t_e and the threads can run more independently and longer.

As we employ load balancing only for large-scale scenarios, we have to investigate weak scaling as well. To this end, we use a fixed number of $10\,k$ particles per core in a test scene similar to the $125\,k$ corner dam break. Again, we compute the relation between serial and parallel execution time. In Fig. 6 (right), results of our weak scaling study are shown. Here, our method outperforms all other investigated methods. In this test case, we use one queue per thread for our model. Some areas in the simulation require more

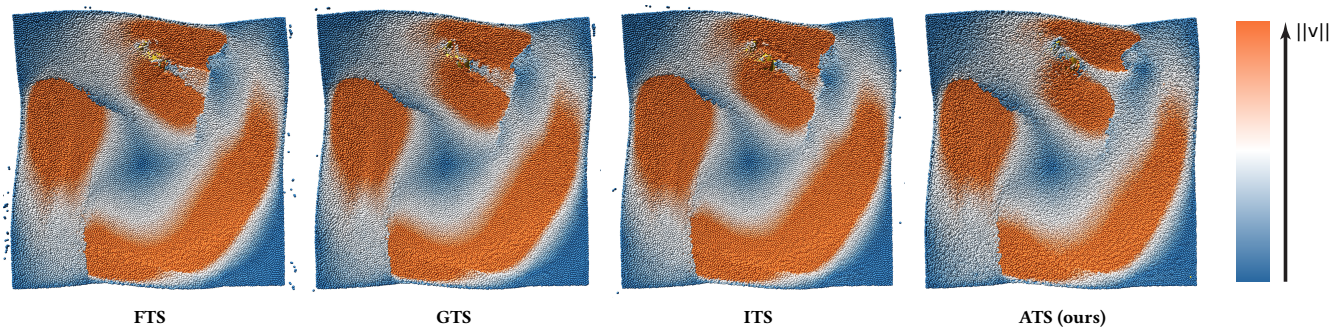


Figure 7: Visual comparison of the methods for the radial flow as presented in Fig. 2. Magnitudes of the velocities are color-coded. We observe only minor differences in the velocity field. However, in the flow behind the collision object, our model leads to slightly damped velocities compared to GTS and ITS.

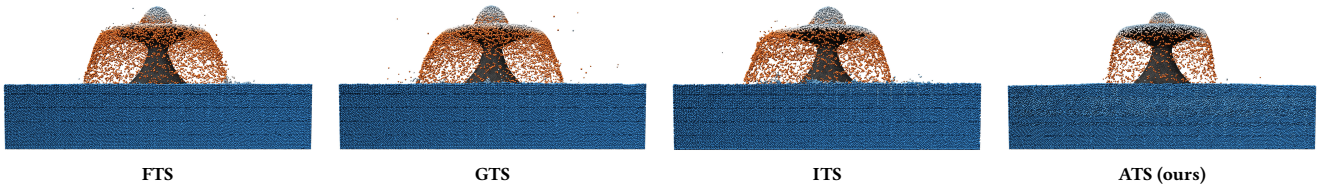


Figure 8: Side-by-side comparison of integration methods applied to the fountain scenario. Magnitudes of the velocities are color-coded using the same color map as in Fig. 7. Due to the velocity damping at the collision object, the fluid falls down steeper in our model compared to FTS, GTS, and ITS.

computation time, as there is more interaction and smaller time steps are needed. Hence, our method introduces waiting threads and the scaling slightly decreases. We address this issue by spawning at least three queues per thread when simulating larger scenarios, such as the 10 M dam break.

In our experiments, the overhead introduced by the queue handling was almost negligible compared to neighborhood search and the particle attribute computation itself. Although we add some extra work, our approach reduces the overall number of operations for the particles, compensating the overhead.

5.2 Visual comparison

We also perform a visual comparison to explore differences of the considered time integration methods. To this end, we compare the resulting animations of the fountain and the radial flow scene.

We observe one noticeable difference between our method compared to other time integration strategies: our asynchronous time integration model dampens the fluid velocities to a small extent, in particular, at interaction with boundary objects. In the radial flow scenario, only small differences in the velocity field can be observed between the tested methods. To identify them, we color the particles by their velocity magnitude, as shown in Fig. 7. Overall, the velocity magnitudes look similar. At the flow behind the collision object, smaller velocities occur in our model compared to FTS. Using ITS or GTS, the velocities are also damped, but not as much as in our model.

The variations get larger in the fountain scenario as shown in Fig. 8. Here, the overall shape behaves considerably different, as it can be seen in the accompanying video. Due to the damped velocity magnitudes in our model, the fluid particles move more slowly than for to FTS and GTS when pouring down from the top of the cone. The fluid on the top also moves less compared to ITS. As in ITS, inactive particles are linearly interpolated and the particles on top allow for large time steps, the shape on top starts to move even more compared to FTS. In both scenes, GTS and FTS look quite similar. However, GTS produces many splashes of single particles. We do not employ explicit shock handling on GTS and, therefore, these shocks produce droplets when the fluid hits the top of the cone and when the waterfall hits the fluid surface.

Apart from the above mentioned differences, our time integration method can equally be used to create visually appealing results. However, there is one important aspect we have to mention: besides stability, we focus on performance and aim to use the largest possible constants λ_V and λ_F for the CFL condition. With ITS and GTS, we observe increased instabilities at some points in the simulation. In the radial flow scenario, this occurs when the fluid hits the collision object, and in the fountain scenario it results in large movements of the fluid on top of the fountain.

5.3 Study on velocity damping

As we observe increased damping of velocity magnitudes at interaction with collision objects with our model, we conduct a quantitative comparison of the fluid velocities. To this end, we investigate

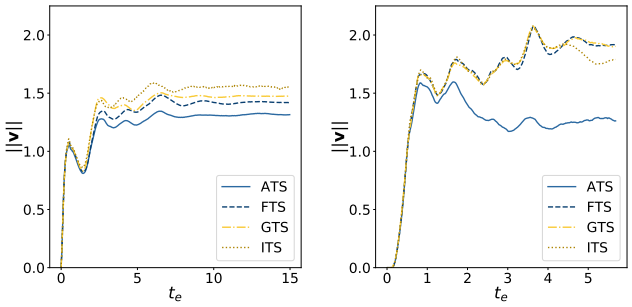


Figure 9: Measurements of our velocity damping study: (left) mean velocities of the radial flow scenario, (right) mean velocities of the radial flow scenario. As we do not observe major differences in the mean of particle velocity magnitudes when considering all particles, we only include particles around the collision objects in our study.

the mean of particle velocity magnitudes over time for the radial flow and the fountain scenario. To examine the variations between the different time stepping models at the interaction, we exclude parts of the simulation in the measurements. In case of the fountain scenario, we neglect the almost resting fluid particles beneath the fountain and in the radial flow scenario, we only consider particles around the bunny. Fig. 9 (left) shows plots of the measurements for the radial flow scenario. Similar to the visual inspection (Sec. 5.2), we recognize only small variations in velocities. Using our time stepping model, velocities are slightly smaller than for FTS, GTS, or ITS. These variations become more apparent in the fountain scenario (Fig. 9, right). In this case, the velocity means are much more damped when interacting with the collision object. This leads to overall smaller velocities and the fluid falls down more steeply.

Additionally, we compare simulations using the time integration scheme of Eq. 4 and the semi-implicit Euler. This test does not reveal any relevant differences in damping of particle velocities. Investigating the influence of the time step size (controlled by the values of the constants $\lambda_{\mathbf{v}}$ and $\lambda_{\mathbf{F}}$) on damping does not indicate any significant change of velocity magnitudes. Our general observation is that damping of velocity magnitudes is related to the combination of our time stepping model and the interaction with static objects.

Detailed results of the study regarding the time stepping models, the influence of the time step size, and the modified integration scheme can be found in the supplementary material.

6 CONCLUSION AND FUTURE WORK

We have presented an asynchronous time integration method for SPH that allows for dedicated time step sizes for each particle and, thus, for increased efficiency of simulations. We do not include any barriers in the simulation and are able to achieve speedups up to a factor of 7.5. Additionally, an efficient concept for parallelization of asynchronous integration for SPH has been presented and tested in terms of absolute runtime and scaling. We have conducted a performance comparison to FTS, GTS, and ITS on five scenarios with different characteristics and also performed a visual comparison of the simulation results to show the specific properties of our model.

There are still possible improvements to our model. If most of the fluid is barely moving, our method still outperforms GTS and ITS, but the speedup decreases. As the work of Goswami and Pajarola [2011] shows promising results for barely active regions, a natural extension would be to follow their concept and handle barely active particles as static ones. Moreover, only small values for $\lambda_{\mathbf{v}}$ and $\lambda_{\mathbf{F}}$ are possible, as we employ only basic shock handling by clamping Δt_i to the minimum of its neighborhood, thus restricting the possible time step size. A more sophisticated shock handling as presented by Ihmsen et al. [2010] could improve stability and open up the possibility to use larger values for $\lambda_{\mathbf{v}}$ and $\lambda_{\mathbf{F}}$. Since we only allow for local constraints based on the compact neighborhood, we would need an equivalent formulation depending only on neighborhood information. For the same reason, it is currently not possible to incorporate iterative SPH models to handle incompressibility constraints. In their original formulation, iterative models such as implicit incompressible SPH [Ihmsen et al. 2014a] or divergence-free SPH [Bender and Koschier 2017], need a global convergence criteria. In this context, we are also looking for a condition to handle compression locally.

Compared to GTS and ITS, our time stepping model leads to higher velocity damping of the fluid when interacting with collision objects. Investigating the combination of our asynchronous time integration with different boundary models would be an interesting topic for future work. Another research direction would be the exploration of how our time stepping model behaves in a massively parallel environment, such as a cluster computer or using GPUs. On the one hand, the simulation domain would need to be split in many more and smaller tasks and we would have to spawn many more queues. Hence, the possibility that neighboring particles are stored in other queues would increase. On the other hand, our threads could run more independently and the particle accessing overhead would be reduced to a minimum; hence, we assume that our approach would be viable for such environments.

ACKNOWLEDGMENTS

This work is partly supported by "Kooperatives Promotionskolleg Digital Media" at Stuttgart Media University and University of Stuttgart.

REFERENCES

B. Adams, M. Pauly, R. Keiser, and L. J. Guibas. 2007. Adaptively sampled particle fluids. *ACM Transactions on Graphics* 26, 3 (2007), 48:1–48:7. DOI: https://doi.org/10.1145/1276377.1276437

N. Akinci, G. Akinci, and M. Teschner. 2013. Versatile surface tension and adhesion for SPH fluids. *ACM Transactions on Graphics* 32, 6 (2013), 182:1–182:8. DOI: https://doi.org/10.1145/2508363.2508395

N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, and M. Teschner. 2012. Versatile rigid-fluid coupling for incompressible SPH. *ACM Transactions on Graphics* 31, 4 (2012), 62:1–62:8. DOI: https://doi.org/10.1145/2185520.2185558

R. Ando, N. Thuerey, and R. Tsuruno. 2012. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Transactions on Visualization and Computer Graphics* 18, 8 (2012), 1202–1214. DOI: https://doi.org/10.1109/TVCG.2012.87

X. Ban, X. Wang, L. He, Y. Zhang, and L. Wang. 2016. Adaptively stepped SPH for fluid animation based on asynchronous time integration. *Neural Computing and Applications* (2016), 1–10. DOI: https://doi.org/10.1007/s00521-016-2286-8

M. Becker and M. Teschner. 2007. Weakly compressible SPH for free surface flows. In *Eurographics/SIGGRAPH Symposium on Computer Animation*. 209–218. DOI: https://doi.org/10.2312/SCA/SCA07/209-218

J. Bender and D. Koschier. 2017. Divergence-free SPH for incompressible and viscous fluids. *IEEE Transactions on Visualization and Computer Graphics* 23, 3 (2017), 1193–1206. DOI: https://doi.org/10.1109/TVCG.2016.2578335

H. Bhattacharya, Y. Gao, and A. W. Bargteil. 2015. A level-set method for skinning animated particle data. *IEEE Transactions on Visualization and Computer Graphics* 21, 3 (2015), 315–327. DOI: https://doi.org/10.1109/TVCG.2014.2362546

M. Desbrun and M.-P. Cani. 1999. *Space-time adaptive simulation of highly deformable substances.* Research Report RR-3829. INRIA.

M. Desbrun and M.-P. Gascuel. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*. 61–76.

P. Goswami and C. Batty. 2014. Regional time stepping for SPH. In *Eurographics 2014 - Short Papers*. 45–48. DOI: https://doi.org/10.2312/egsh.20141011

P. Goswami and R. Pajarola. 2011. Time adaptive approximate SPH. In *Proceedings of Workshop on Virtual Reality Interaction and Physical Simulation*. 19–28. DOI: https://doi.org/10.2312/PE/vriphys/vriphys11/019-028

S. Grottel, M. Krone, C. Müller, G. Reina, and T. Ertl. 2015. MegaMol — Prototyping framework for particle-based visualization. *IEEE Transactions on Visualization and Computer Graphics* 21, 2 (2015), 201–214. DOI: https://doi.org/10.1109/TVCG.2014.2350479

D. Harmon, E. Vouga, B. Smith, R. Tamstorf, and E. Grinspun. 2009. Asynchronous contact mechanics. *ACM Transactions on Graphics* 28, 3 (2009), 87:1–87:12. DOI: https://doi.org/10.1145/1531326.1531393

X. He, H. Wang, F. Zhang, H. Wang, G. Wang, and K. Zhou. 2014. Robust simulation of sparsely sampled thin features in SPH-based free surface flows. *ACM Transactions on Graphics* 34, 1 (2014), 7:1–7:9. DOI: https://doi.org/10.1145/2682630

C. J. Horvath and B. Solenthaler. 2013. *Mass preserving multi-scale SPH.* Technical Memo #13-04. Pixar.

M. Huber, S. Reinhardt, D. Weiskopf, and B. Eberhardt. 2015. Evaluation of surface tension models for SPH-based fluid animations using a benchmark test. In *Proceedings of Workshop on Virtual Reality Interaction and Physical Simulation*. 41–50. DOI: https://doi.org/10.2312/vriphys.20151333

M. Ihmsen, N. Akinci, M. Becker, and M. Teschner. 2011. A parallel SPH implementation on multi-core CPUs. *Computer Graphics Forum* 30, 1 (2011), 99–112. DOI: https://doi.org/10.1111/j.1467-8659.2010.01832.x

M. Ihmsen, N. Akinci, M. Gissler, and M. Teschner. 2010. Boundary handling and adaptive time-stepping for PCISPH. In *Proceedings of Workshop in Virtual Reality Interactions and Physical Simulation*. 79–88. DOI: https://doi.org/10.2312/PE/vriphys/vriphys10/079-088

M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner. 2014a. Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics* 20, 3 (2014), 426–435. DOI: https://doi.org/10.1109/TVCG.2013.105

M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner. 2014b. SPH fluids in computer graphics. In *EG 2014 - STARs*. 21–42. DOI: https://doi.org/10.2312/egst.20141034

K. G. Kale and A. J. Lew. 2007. Parallel asynchronous variational integrators. *International Journal for Numerical Methods in Engineering* 70, 3 (2007), 291–321. DOI: https://doi.org/10.1002/nme.1880

A. Lew, J. E. Marsden, M. Ortiz, and M. West. 2004. Variational time integrators. *International Journal for Numerical Methods in Engineering* 60, 1 (2004), 153–212. DOI: https://doi.org/10.1002/nme.958

P.-L. Manteaux, C. Wojtan, R. Narain, S. Redon, F. Faure, and M.-P. Cani. 2016. Adaptive physically based models in computer graphics. *Computer Graphics Forum* (2016). DOI: https://doi.org/10.1111/cgf.12941

J. J. Monaghan. 1992. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics* 30, 1 (1992), 543–574. DOI: https://doi.org/10.1146/annurev.aa.30.090192.002551

M. Müller, D. Charypar, and M. Gross. 2003. Particle-based fluid simulation for interactive applications. In *Eurographics/SIGGRAPH Symposium on Computer Animation*. 154–159. DOI: https://doi.org/10.2312/SCA03/154-159

J. Orthmann and A. Kolb. 2012. Temporal blending for adaptive SPH. *Computer Graphics Forum* 31, 8 (2012), 2436–2449. DOI: https://doi.org/10.1111/j.1467-8659.2012.03186.x

S. Reinhardt, M. Huber, O. Dumitrescu, M. Krone, B. Eberhardt, and D. Weiskopf. 2017. Visual debugging of SPH simulations. In *21th International Conference Information Visualisation*. (to appear).

C. Schroeder, N. Kwatra, W. Zheng, and R. Fedkiw. 2011. Asynchronous evolution for fully-implicit and semi-implicit time integration. *Computer Graphics Forum* 30, 7 (2011), 1983–1992. DOI: https://doi.org/10.1111/j.1467-8659.2011.02046.x

B. Solenthaler and M. Gross. 2011. Two-scale particle simulation. *ACM Transactions on Graphics* 30, 4 (2011), 81:1–81:8. DOI: https://doi.org/10.1145/2010324.1964976

F. Thaler, B. Solenthaler, and M. Gross. 2014. A parallel architecture for IISPH fluids. In *Proceedings of Workshop on Virtual Reality Interaction and Physical Simulation*. 119–124. DOI: https://doi.org/10.2312/vriphys.20141230

B. Thomaszewski, S. Pabst, and W. Straßer. 2008. Asynchronous cloth simulation. In *Proceedings of Computer Graphics International*.

D. Zhao, Y. Li, and J. Barbič. 2016. Asynchronous implicit backward Euler integration. In *Eurographics/SIGGRAPH Symposium on Computer Animation*. 1–9. DOI: https://doi.org/10.2312/sca.20161217