Adaptive Particle Splitting Based on Turbulence Energy for Fluid Simulations on GPUs

Arno in Wolde Lübke[1,a)] Makoto Fujisawa[2,b)] Taketomi Takafumi[1,c)] Goshiro Yamamoto[1,d)] Jun Miyazaki[3,e)] Hirokazu Kato[1,f)]

Abstract: We present a simulation method for particle-based fluids to capture small-scale features such as splashes in higher resolution than the underlying base simulation. For this purpose, we split particles in visually important regions of the fluid into smaller, high-resolution particles. To reduce the computational overhead introduced by the additional simulation scale, we propose splitting only those particles found within turbulent surface regions that are visible to the camera. We extract these particles in screen space and decide to split them based on their turbulent energy. Further, to compensate for irregularities in the quantity field that are introduced by transitioning particles, we use a simple blending approach to maintain stability. We fully implemented our method for graphics processing units to further accelerate the computational speed. In early experiments we could achieve speed increases of up to two over a high-resolution simulation while preserving similar visual qualities.

1. Introduction

Fluids such as liquids and gases are important visual elements in movies, video games and advertisements. However, to achieve visually pleasing results, simulations are often carried out at high resolution. Additionally, the underlying model of fluids and its numerical solutions requires high computational costs, which make physics-based fluid simulations impractical for real-time applications and lead to long design cycles in offline applications. To alleviate these computational requirements, this study proposes a two-scale approach for particle-based fluids and discusses its implementation on graphics processing units for further performance gains.

Recently particle-based fluids have evolved as an interesting alternative to their grid-based counterparts since their governing equations' concept is more simple and computations are only carried out for each fluid element instead of an embedding data structure. In addition, particle-based methods trivially conserve mass.

For the purpose of solving the particle-based model, Smoothed Particle Hydrodynamics (SPH) has become a commonly used method that allows the reconstruction of continuous quantity fields by taking a weighted average of the neighboring particles' values. Using SPH, real-time results have been accomplished [15]. However, the number of particles is limited to a couple of thousands for real-time applications. In addition, small-scale details such as splashes cannot be captured accurately, and the final visual result of the fluid appears to be blobby.

To simulate more particles in shorter time, graphics processing units (GPUs), which posses multiple processing units for parallel computations, have been used to speed-up the computation of SPH Fluids. Speed-ups on order of ten have been reported in early results [10]. Furthermore, multi-resolution methods that simulate fluid areas at different scales depending on certain criteria have been proposed and succeeded in preserving visual detail while significantly reducing the computational cost [21].

To our knowledge, only a few efforts have been made to combine GPUs and multi-resolution SPH fluids. Furthermore, existing implementations produced no considerable performance gains if the fluid was very turbulent [17]. This study proposes an easy-to-implement multi-scale simulation method for graphics processing units that splits particles in visually important regions of the fluid into smaller, high resolution particles. To reduce the computational overhead introduced by the additional simulation scale, particles are only split in turbulent surface regions that are visible to the camera. For this purpose, surface particles are extracted in screen space and divided based on their turbulent energy.

The contributions of this study may be summarized as follows:
- In order to further reduce the computational overhead, this paper proposes an additional splitting criterion based on the turbulent energy of a particle, which ensures that surface particles are only split in turbulent, and visually important areas of the fluid.
- Recently, the concept of blending was introduced to multi-

[1] Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara 630-0192, Japan
[2] University of Tsukuba, 1-2 Kasuga, Tsukuba, Ibaraki 305-8550, Japan
[3] Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8552, Japan
[a)] arno-w@is.naist.jp
[b)] fujis@slis.tsukuba.ac.jp
[c)] takafumi-t@is.naist.jp
[d)] goshiro@is.naist.jp
[e)] miyazaki@cs.titech.ac.jp
[f)] kato@is.naist.jp

resolution fluids in order to maintain stability in areas where particles are transitioning between high and low resolutions. In this study, this model is simplified to reduce the overhead that comes with the original method. In essence, in order to avoid numerical issues that come with this simplifications, this study restricts the movement of a particle based on the Courant Friedrichs Lewy condition.

- Since merging of particles is one of the most difficult operations to implement on GPUs and is not comprehensively described in current literature, an easy to implement merging method for GPUs is proposed.

2. Related Work

Algorithms for fluid animation in computer graphics rely heavily on findings from the research field of fluid mechanics. In particular, the motion of a fluid can be formulated by a set of partial differential equations, the so called Navier-Stokes Equations. These equations were formulated independently by Claude Navier and George Stokes in 1822 and 1845, respectively.

Particle systems, as introduced to computer graphics by Reeves [19], are a popular model to describe natural phenomena that are difficult to capture using classical surface representations such as liquids, smoke or clouds.

A numerical method that is commonly used in combination with a particle system to model fluid behaviour is Smoothed Particle Hydrodynamics (SPH) [14].

Müller et. al. [15] demonstrated that SPH can be used in real-time fluid simulations using specially designed weighting kernels and the ideal gas equation to relate densities to pressures. This method has the disadvantage that the fluid is highly compressible. Becker and Teschner [2] popularized a weakly compressible SPH model for computer animation. Furthermore, the predictive corrective incompressible SPH (PCISPH) method [20] is a modern SPH fluid solver that allows for incompressibility at high simulation time steps. Recently, Maklin and Müller [13] presented a solver based on the Position Based Dynamics framework [16], that increases the integration time-step by ten over the PCISPH method.

Bridson et. al. [3] suggested a way to create divergence free, turbulent velocity fields for procedural animation by taking the curl of a noise function. Kim et. al. [11] used this result in combination with wavelet noise [4] to add small scale detail as a post-processing step. Fujisawa et. al. [6] applied these results to the SPH fluid model by formulating a wavelet decomposition for particle-based fluids and adding details using vortex sub-particles.

The goal of this study is to present methods that can accelerate these approaches to fluid simulation. Related research can generally be divided into three different approaches:

Multi-Resolution Fluids Multi-resolution approaches aim to allocate more computational resources to specially-defined fluid regions to increase the visual quality and/or physical accuracy of the simulation.

Adams et. al [1] sample the fluid using the geometric local feature size as criterion so that complex geometric regions are represented by more particles than less complex areas. In addition to geometric criteria, this study investigates the use of a physics-related property, the turbulence energy of a particle as a criterium to avoid costs that arise due to splitting and merging particles. Further, this study suggests a new way to extract surface particles by exploiting the popular screen space fluid rendering algorithm of van der Laan et. al. [22]. Zhang et. al. [23] explain how adaptive sampling can be mapped to the GPU exploiting the rendering pipeline. Orthmann and Kolb [17] present a method that allows a smooth transition between particles of different resolutions avoiding numerical problems that might occur due to poor resampling of the fluid. This study makes use of this method but simplifies it in order reduce computational overhead and ease its implementation.

Graphics Processing Units The advent of programmable rendering pipelines in computer graphics gave researchers the opportunity to port various algorithms to the graphics processing unit (GPU) in order to benefit from its parallel architecture. Harada et. al. [10] implemented the SPH fluid model on the GPU using OpenGL and C for Graphics. They reported that their method enabled them to use ten times as many particles as previous real-time simulations.

NVIDIA's white paper on particles [9] showed an efficient way to compute a particle's neighborhood on GPU's and is commonly used in GPU implementations of SPH fluids. Recently, Goswami et. al. [8] made use of z-ordering to arrange particles for neighborhood queries and access particle data through fast shared memory.

This study follows the approach of [9] and shows a way to implement the splitting and merging operations of multi-resolution fluids.

Approximation In order to save computational time, approximation strategies try to avoid computations for particles from which contributions to the whole system are neglectable.

Goswami and Pajarola [7] classify fluid particles as either active or inactive. Particles are considered inactive if they do not contribute to the visual quality and their speed is below a certain threshold. Inactive particles are then not considered in the SPH computations.

Pelfrey and House [18] optimize the standard SPH approach by computing the neighborhood of a particle only when necessary.

3. Particle-Based Fluid Simulation

This section presents the model for particle-based fluids employed in this paper that is used for comparison in the results section and extended in the subsequent section.

Particle-based fluid simulations employ a particle system to discretize the fluid. In this work, the motion of a fluid particle i of the system is governed by

$$\mathbf{a}_i = \frac{1}{\rho_i}(\mathbf{f}_i^p + \mathbf{f}_i^\mu + \rho_i \mathbf{g}) = \frac{1}{\rho_i}\Big(- (\nabla p)_i + \mu(\Delta \mathbf{v})_i + \rho_i \mathbf{g}\Big). \quad (1)$$

in accordance to the incompressible Lagrangian Navier-Stokes equations. Equation 1 relates the acceleration of the particle to the pressure, viscosity and external forces it undergoes, where μ

denotes the dynamic viscosity of the fluid, ρ_i is the density of the fluid at the particle position, p is the pressure field, and $\mathbf{v}$ is the velocity field of the fluid.

In order to evaluate the forces acting on a particle i in the model equation, a Smoothed Particle Hydrodynamics approach is used that approximates the pressure and viscosity force as follows:

$$\mathbf{f}_i^{\mathrm{p}} = -\rho_i \sum_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) m_j \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{2}$$

$$\mathbf{f}_i^{\mu} = \mu \sum_j (\mathbf{v}_j - \mathbf{v}_i) \frac{m_j}{\rho_j} \Delta W(\mathbf{x}_i - \mathbf{x}_j, h). \tag{3}$$

where the index j references all particles that lie within an effective radius h around particle i and W is a function (kernel) that weighs the contribution of particle j to the net force acting on particle i. In this paper the kernels of Müller et. al. [15] are used.

The density of a particle i is computed by the standard SPH estimator,

$$\rho_i = \sum_j m_j W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{4}$$

and its pressure is given by Tait's equation

$$p_i = B\left(\left(\frac{\rho_i}{\rho_0} \right)^{\gamma} - 1 \right) \tag{5}$$

where γ is chosen to be one, B is Tait's coefficient and ρ_0 is the rest density of the fluid.

Boundary handling is implemented a suggested by Becker and Teschner [2] using boundary particles for solid obstacles and cohesion forces to model surface tension effects.

Eventually, the particle positions at the next time step are optained by integrating Equation 1 in time applying Euler's method.

4. Adaptive Particle Splitting

The previous section presented a simulation method for one simulation scale. This section proposes a method that supports two simulation scales, a high-resolution scale to better capture small-scale details and a low-resolution scale to save computational time in uninteresting regions of the fluid. For this purpose, in the following, the two-scale extension of the previous method is reviewed, before a way to dynamically change resolutions within the fluids is discussed. Finally, the criteria for resampling the fluid in this work are given.

4.1 Two-Scale Particle Model

Similar to Solenthaler et. al. [21], this work restricts itself to using only two different resolutions. The benefits of this approach are fewer errors due to numerical approximations, less implementation costs and the observation that it is visually-sufficient to describe fluid regions either with high-resolution particles or low-resolution particles. Further, this paper covers only the case wherein the resolution is double for small-scale details. However, the following discussion might easily be extended for different increases in resolution.

As a result of the additional simulation scale, a distinction must be made between the mass and effective radius of a low- and high-resolution particle. Both attributes relate to each other as follows: $m_L = 8m_H, h_L = 2h_H$. Different effective radii require changes to the definition of a particles neighborhood to maintain mutual visibility for all particle pairs. This paper follows work of Desbrun and Canny [5], and includes a particle j in the SPH computations for particle i, if $\|\mathbf{x}_i - \mathbf{x}_j\| \leq \max(h_i, h_j)$ holds, where h_i and h_j denote the particle radius of particle i and j.

Due to the adjusted neighbor definition and to obey Newton's third law, one has to ensure that the forces for a particle pair of different resolutions cancel each other. In practice [1], a common method to achieve this for two particles i and j, is to take the arithmetic mean of their smoothing kernels evaluated with their distance and respective effective radius to weight specific quantities. For instance, the SPH term to compute the pressure force acting on a particle i is rewritten as

$$\mathbf{f}_i^{p} = -\rho_i \sum_{j \neq i} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \frac{\nabla W_{ij}^{h_i} + \nabla W_{ij}^{h_j}}{2}. \tag{6}$$

Forces due to viscosity, boundaries and surfaces tension as well as the SPH density estimator are adjusted similarly.

4.2 Particle Resampling

Replacing particles of different resolutions can lead to poor positioning of the newly inserted particles which results in high density fluctuations within the fluid. In these situations, the arising high pressure forces require small simulation time-steps to ensure stability. To maintain large time-steps, recently Orthmann and Kolb [17] introduced the concept of blending to the field of particle-based fluids. The following subsection describes an alternative to this method, by limiting a particle's motion based on the Courant Friedrichs Lewy (CFL) condition, in order to omit parts of the computational overhead introduced by the original method. Subsequently, particle splitting and merging are described.

4.2.1 Particle Blending

Particle blending lets particles of different resolutions that are about to be exchanged to coexist for a certain time period. During this time period the contribution to the system of newly inserted particles is steadily increased whereas the contribution of the particles that were supposed to be replaced is decreased. Finally, if the contribution of a particle is zero, it is removed from the system. This way, the potential high pressure forces introduced by the new particles are attenuated over a certain period of time allowing the system to reach to a stable configuration before particles are finally replaced. The process of particle blending is illustrated in **Fig. 1**.

For implementing particle blending, each particle is assigned a state s. A particle is in `insert` state if it is about to be inserted to the system, in `delete` state if it is about to be replaced or in `default` state if it is neither being inserted nor being replaced. Further, each particle has a blending coefficient Λ that determines its contribution to the system. Each time step, the blending coefficient of a particle i is updated depending on its state in the following way:

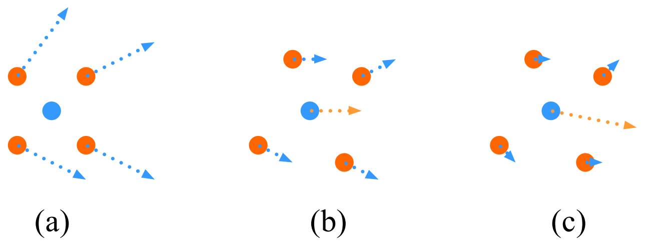


Fig. 1: Particle Blending: (a) shows the situation when four new high-resolution particles (orange) where just inserted. All particles have a blending coefficient of zero and, therefore, do not contribute to the system. The low resolution particle (blue), that is about to be deleted, has still a blending coefficient of one and therefore fully contributes to the system (blue arrow). (b) shows the situation after a short period of time has elapsed: The low-resolution particle's force is attenuated, whereas the high-resolution particles' contribution (blue arrows) has increased. (c) shows the situation shortly before the low-resolution particle is being removed. Its exerted force is strongly attenuated, whereas the high-resolution particles' contribution to the system is almost one hundred percent.

$$\Lambda_i^{n+1} = \Lambda_i^n + \begin{cases} \Delta\Lambda & \text{if} s_i = \texttt{insert} \\ -\Delta\Lambda & \text{if} s_i = \texttt{delete} \\ 0 & \text{if} s_i = \texttt{default} \end{cases} \tag{7}$$

where $\Delta\Lambda$ is a constant blending increment. For a newly inserted particle the blending coefficient is set to zero (meaning it has no contribution to the system) and increased to one (meaning it contributes fully to the system) using the equation above. Likewise the blend coefficient for a particle that is supposed to be replaced starts at one and is decreased to zero, before it is finally removed from the system.

Eventually, the blending coefficient is used to weigh the force that a particle exerts on another particle. For instance, the pressure force acting on particle i may now be computed by

$$\mathbf{f}_i^p = -\rho_i \sum_{j\neq i} \Lambda_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \frac{\nabla W_{ij}^{h_i} + \nabla W_{ij}^{h_j}}{2}. \tag{8}$$

taking into account the contributions of the neighboring particles j to the system.

Unfortunately, even with the use of particle blending in some cases the system may experience instabilities due to inaccurate sampling. In order to make the simulation more robust, this work restricts the motion of a particle according to the CFL condition which is a necessary condition for stability. The CFL condition basically states that per time step Δt information should not be propagated further than the spatial increment Δx of the underlying grid, that is, in one dimension, for a uniform grid $\frac{u\Delta t}{\Delta x} \leq 1$, where u denotes the maximum velocity throughout the domain, must hold. Translating this condition to a particle-based discretization of the simulation domain, this work sets the spatial increment Δx proportional to the effective radius of the simulation h, which results in the following constraint for the velocity of a particle

$$||\mathbf{v}|| \leq \frac{\alpha h}{\Delta t} \tag{9}$$

with some constant α. Consequently, in the integration step of the proposed method, the computed velocity $\mathbf{v}_i$ for a particle i is adjusted as follows

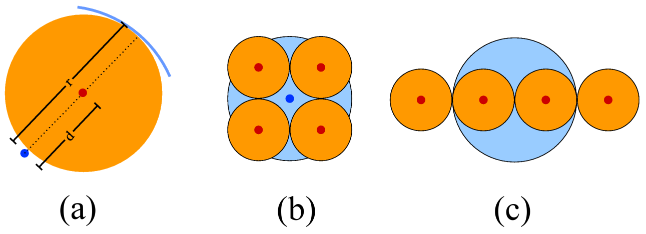


Fig. 2: Splitting and merging: (a) shows the initial distance of a high-resolution particle to its parent low-resolution particle (here $d \approx \frac{1}{2} r_i$). (b) shows how high-resolution particles are aligned when being inserted. It also shows a valid configuration for high-resolution particles to merge into a single low-resolution particle. (c) shows an undesired configuration for high-resolution particles to merge.

$$\mathbf{v}_i^* = \min\left(||\mathbf{v}_i||, \alpha \frac{h}{\Delta t} \right) \frac{\mathbf{v}_i}{||\mathbf{v}_i||} \tag{10}$$

in order to ensure this constraint given by Equation 9.

4.2.2 Particle Splitting

When a low-resolution particle i is determined to be split, eight new high resolution particles are inserted to the system. These particles are positioned on an axis-aligned cube around the low resolution particle, as illustrated by Figure **Fig. 2** (b). The distance of each high resolution particle to the low resolution particle should be proportional to the volume the low resolution particle occupies (see Fig. 2 (a)), and is computed as follows:

$$d \propto \frac{1}{2} r_i = \frac{1}{2} \left(\frac{3V_i}{4\pi} \right)^{\frac{1}{3}} = \frac{1}{2} \left(\frac{3m_L}{4\pi\rho_i} \right)^{\frac{1}{3}} \tag{11}$$

The velocity of all newly inserted particles equals the velocity of the low resolution particle. Finally, the low resolution particle's state is set to `delete` and the states of the high-resolution particles are set to `insert`.

4.2.3 Particle Merging

Merging saves computational time in regions of the fluid that do not require high spatial resolutions. If eight high-resolution particles, which lie close to each other are part of these regions they should be combined to one low resolution particle. Position and velocity of this particle are computed as the average over the high resolution particles that are being merged.

It should be noted that in addition to being close to each other, the particles to be merged should be aligned in a way that they fill out the volume of the to-be-inserted low-resolution particle as shown in Fig. 2 (b) + (c). This constraint is important to improve the sampling quality. A way of identifying high-resolution particles that can be merged for GPU implementations is described in Section 5.2.

4.3 Resampling Criteria

Here the criteria for dynamically changing the simulation scale in a fluid are given. This work aims to keep the visible surface particles at high resolutions, while the rest of the fluid is animated by low-resolution particles. To further reduce splitting and merging operations and keep the particle count low, splitting operations are restricted by the turbulent energy of a particle, while merging is enforced for particles with low turbulent energy.

4.3.1 Surface Areas

As only surface areas of the fluid are directly visible to the camera, they comprise a desirable region of the fluid to be sampled at higher resolutions. To extract this area in previous research normally the distance of a particle to the center of mass of its neighbors is computed [21]. If this distance is higher than a user-defined threshold, the particle is recognized to be a surface particle. However, this method does not always select surface particles correctly, especially if the fluid is compressible, which may be desirable in order to achieve higher integration time-steps. Moreover, this method also detects, surface particles that might not directly be visible to the camera, e.g. particles at the back of a fluid or particles outside the view frustum of the camera. For this reason, this study extracts surface particles in screen space, while rendering using a screen space rendering algorithm.

4.3.2 Turbulence Energy

To identify turbulent regions within the fluid, this paper uses a wavelet-based approach similar to [11]. Wavelets are a mathematical tool to hierarchically decompose a function into a lower resolution part of the function itself and the details that are lost due to the decrease in resolution. The latter part is known as detail coefficients. Applied to the velocity field of a fluid, the detail coefficients may be seen as a measure of the local differences of a certain velocity component and are therefore a suitable criterion for turbulence. Unfortunately, the discrete wavelet decomposition can only be applied to grids. For this reason particle data needs to be projected onto grid cells. However, this would cause additional numerical diffusion and impose more strict memory and computational requirements. To avoid these issues this paper follows the approach of Fujisawa et. al. [6] who define a discrete wavelet transform for particles as follows,

$$\hat{u}_i = \frac{1}{\sqrt{s}\psi_{sum}} \sum_j u_j \psi((\mathbf{x}_i - \mathbf{x}_j)/s) \tag{12}$$

where $\hat{u}_i$ is the wavelet transform of the first velocity component of particle i, ψ is the Mexican hat mother wavelet, s is the wavelet scale and $\psi_{sum} = \sum_j \psi_{sum}((\mathbf{x}_i - \mathbf{x}_j)/s)$ is a norm to address undesirable effects at the boundary of the fluid. Given the wavelet transform of the velocity $\hat{\mathbf{v}}_i = (\hat{u}_i, \hat{v}_i, \hat{w}_i)$ of a particle i, its turbulent energy is then defined as

$$E_i = \frac{1}{2}||\hat{\mathbf{v}}_i||^2. \tag{13}$$

Given the energy distribution of the fluid, a low-resolution particle is then determined to be split if its energy exceeds a certain threshhold E^θ. In the same way, a high-resolution particle is marked for merging if its energy is below E^θ.

5. Implementation

The two-scale method presented in this paper was implemented fully on a NVIDIA GeForce GTX 670 GPU using CUDA 5.0. To increase memory coherence and avoid cache misses particle data is stored as a structure of arrays and reordered according to its position at the beginning of each time-step. For fast neighbor lookup the method proposed by Green [9] is adopted, which partitions the simulation space using a uniform grid with an spatial increment of h and sorts all particles according to their grid indices. However, due to the additional simulation scale two grids are used to minimize the number of considered particles when searching for neighbors.

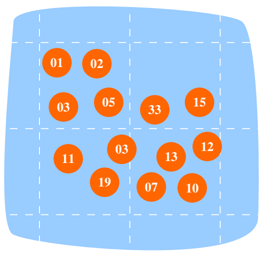
```
while animating do
    compute particle neighborhoods (cf. Sec. 5)
    foreach particle $i \in A_L^n \cup A_H^n$ pardo
        compute density $\rho_i$ (cf. Eq. 8)
    end
    foreach particle $i \in A_L^n \cup A_H^n$ pardo
        compute acceleration (cf. Eq. 8)
        compute energy (Eq. 13)
    end
    foreach particle $i \in A_L^n \cup A_H^n$ pardo
        update position and velocity (cf. Eq.10)
        update blending coefficient (Eq. 7)
        if $\Lambda_i = 1$ then
            $s_i \leftarrow$ default;
        end
        if $\Lambda_i > 0$ then
            add($A_i^{n+1}, i$);
        end
    end
    foreach particle $i \in A_L^n$ pardo
        if $s_i$ = default $\wedge$ split(i) (cf. Sec. 4.3) then
            insert($A_H^{n+1}$, i); (cf. Sec. 4.2.2)
            $s_i \leftarrow$ delete;
        end
    end
    foreach particle $i \in A_H^n$ pardo
        merge(i); (cf. Sec. 5.2)
    end
end
```

Algorithm 1: Summary of the proposed method, A_L^n is a list of all active low- resolution particle ids at the current time step and A_H^n is its high-resolution counterpart. A_i^{n+1} refers to the list of resolution of particle i at the next time step.

The method presented and its implementation are depicted in Algorithm 1. The following is a short discussion of the GPU implementation of splitting and merging in this work. To keep track of all active particles of the two different scales, extra arrays are employed, which are denoted by A_L for the low resolution and A_H for the high resolution. If a particle i should be deleted from the system, its id is simply not added to its active list at the next time step A_i^{n+1} at the end of the simulation step.

5.1 Splitting

The GPU implementation of splitting in this work is straightforward and essentially follows the procedure described in 4.2.2. Each thread checks if its particle can be split. If so, eight high-resolution particles are inserted to A_H^{n+1} and initialized. During initialization these particles are positioned around a cube that is centered around the low-resolution particle. The velocity is simply inherited from the parent and their blending coefficient is set to zero. Lastly, the states of the high-resolution particles are set to `insert` and the state of the low-resolution particle is set to `delete`.



numParticles	cellOwner	newLowId
⋮	⋮	⋮
4	1	$\lvert A_L^n \rvert + 1$
2	15	-1
⋮	⋮	⋮
3	3	-1
4	7	$\lvert A_L^n \rvert + 2$
⋮	⋮	⋮

Fig. 3: Detecting particles for merging in two dimensions: The domain is subdivided by a grid. For each grid cell the amount of particles that can be merged is counted. Then one high-resolution particle to be responsible for the cell is chosen and, if a cell contains four mergable particles, the id of the to be inserted low-resolution particle is assigned to the cell.

5.2 Merging

The implementation proposed in this work uses a uniform grid to separate the simulation domain in order to find high-resolution particles that can be merged (see **Fig. 3**). Examining the simulation domain per grid cell has the advantage that particles within the cell are likely to be arranged properly for merging. Furthermore, it is easier to search for high-resolution particles in a grid cell as compared to checking the neighbors of each. Since exactly eight particles have to be merged, the spatial increment of the grid Δx is chosen depending on the volume that eight high-resolution particles usually occupy, that is $\Delta x \propto (8V_H)^{\frac{1}{3}}$ where $V_H = m_h/\rho_0$ is the volume a high-resolution particle occupies when it is at rest.

In case eight particles that are eligible to merge are found for a grid cell, one particle (thread) is delegated to add the index of the to-be-inserted particle to the array that stores all active low-resolution particles A_L^{n+1}. For this purpose, three arrays of the size of the number of grid cells are needed: one to store the number of particles that can be merged per cell, one to save the id of the particle that is delegated to the cell and one to store the id of the to-be-inserted low-resolution particle per cell. These arrays are referred as to `numParticles`, `cellOwner` and `newLowID` in Fig. 3.

For every grid cell that contains high-resolution particles, the actual merging process is carried out in three steps, which can be briefly described as follows

(1) Count all particles that should be merged, and determine the `cellOwner`.
(2) If merging is possible, create a new low-resolution particle and determine its index in A_L^{n+1} using the `cellOwner`.
(3) If a new particle is created, initialize its position and velocity by taking the average over all eight high-resolution particles.

In all steps atomic operators had to be used to avoid race conditions.

6. Results

To evaluate the ideas of the previous sections, the proposed algorithm is compared to the standard SPH algorithm of Section 3

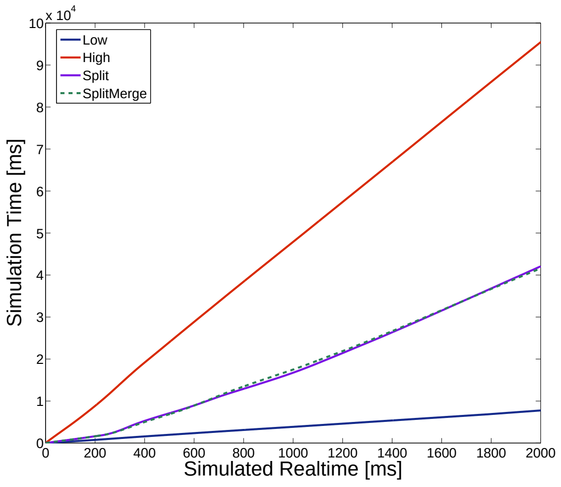


Fig. 4: Simulation timings

Table 1: Physical constants chosen for all test scenarios.

Constant	Symbol	Value
Gravitational Acceleration	$\mathbf{g}$	$9.81\ m/s^2$
Rest Density	ρ_0	$1000\ kg/m^3$
Viscosity	μ	$5.0 \cdot 10^{-3}\ kg/s(\cdot m)$
Tait Coefficient	B	$30\ Pa$
Average particle neighbors	$\bar{n}$	35
Blend Increment	$\Delta\Lambda$	0.02
Energy Threshhold	E^θ	$300J$

in a dam break scenario. To increase the turbulent regions during the simulation an additional pillar has been added to the scene. Four different scenarios have been evaluated: In the first scenario the fluid is animated with the standard SPH approach and 35k fluid particles (low-resolution simulation), whereas in the second scenario the fluid is animated with 280k particles (high-resolution simulation). In the third scenario the simuliation starts with 35k particles, but the fluid is allowed to split the particles. The forth scenario also allows for the merging of particles as well.

The computational time it took to simulate the fluid for a certain period of real time is given in **Fig. 4**. The y-axis denotes the computational time needed for a certain point in real time. As can be seen from the graph, the amount of computational time needed for the low-resolution simulation is about nine seconds, whereas the high-resolution simulations needs about 95 seconds. Both multi-resolution simulations take about the same time: 41 seconds. In total, $2s$ in real-time were animated. For all simulations a time-step $\Delta t = 1\ ms$ was chosen except for the low-resolution simulation where a time-step $\Delta t = 2\ ms$ was chosen. In scenario three, the number of low-resolution particles at end of the simulation decreased to 28k, whereas the number of high-resolution particles is increased to 65k. As for scenerio four, due to successful merging operations, the number of low-resolution particles at the end of the simulation is 31k, whereas the number of high-resolution particles is 38.5k.

7. Conclusion and Future Work

In summary, this work proposed a two-scale particle-based fluid simulation method. The goal was to achieve similar visual qualities of a high resolution simulation by allocating more computional resources to visually important parts of a low-resolution fluid while saving computational time for the rest of the fluid. To achieve this, visually important areas were identified as surface areas with high turbulent energy using the wavelet decomposition of the velocities of the fluid. In order to split and merge particles while still preserving large time-steps a simple blending-based method that trivially maps to graphics processing units was proposed. The experiment conducted in this work suggested that two times faster simulation times are possible while still preserving similar visual qualities, which can lessen the duration of design cycles as well bring more visual fidelity to real-time applications.

As a major downside, the considerable increase of memory due to auxiliary structures for the extra resolution and the blending process should be mentioned. This is especially severe if a high amount of the simulation domain is not needed due to evironmental restrictions. In order to address this problem the usage of more memory efficient data structures such as octrees and their implementation on GPUs as described in [12] can be investigated in future research. Furthermore, the proposed method has been tested for one particular scenario. Different experiments with varying scene complexities should be carried out to find out where the proposed ideas are applicable to get performance gains. It would also be interesting to see if the ideas presented here can be applied to the recently published and very popular method of Macklin and Müller [13].

References

[1] Adams, B., Pauly, M., Keiser, R. and Guibas, L. J.: Adaptively sampled particle fluids, *ACM Trans. Graph.*, Vol. 26, No. 3 (online), DOI: 10.1145/1276377.1276437 (2007).

[2] Becker, M. and Teschner, M.: Weakly compressible SPH for free surface flows, *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp. 209–217 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1272690.1272719⟩ (2007).

[3] Bridson, R., Houriham, J. and Nordenstam, M.: Curl-noise for procedural fluid flow, *ACM Trans. Graph.*, Vol. 26, No. 3 (online), DOI: 10.1145/1276377.1276435 (2007).

[4] Cook, R. L. and DeRose, T.: Wavelet noise, *ACM Trans. Graph.*, Vol. 24, No. 3, pp. 803–811 (online), DOI: 10.1145/1073204.1073264 (2005).

[5] Desbrun, M. and Cani, M.-P.: Space-Time Adaptive Simulation of Highly Deformable Substances, Rapport de recherche RR-3829, INRIA (1999).

[6] Fujisawa, M., Mimura, G., Amano, T., Miyazaki, J. and Kato, H.: A Fast Simulation Method Using SPH and Wavelet for Sub-Particle-Scale Turbulent Flow, pp. 67–72 (online), DOI: 10.2312/PE/PG/PG2011short/067-072 (2011).

[7] Goswami, P. and Pajarola, R.: Time adaptive approximate SPH, *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS* (Bender, J., Erleben, K. and Galin, E., eds.), VRIPHYS 2011, Eurographics, pp. 19–28 (online), available from ⟨http://www.zora.uzh.ch/55770/⟩ (2011).

[8] Goswami, P., Schlegel, P., Solenthaler, B. and Pajarola, R.: Interactive SPH simulation and rendering on the GPU, *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp. 55–64 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1921427.1921437⟩ (2010).

[9] Green, S.: Cuda particles, *nVidia Whitepaper*, Vol. 2, No. 3.2, p. 1 (2008).

[10] Harada, T., Koshizuka, S. and Kawaguchi, Y.: Smoothed particle hydrodynamics on GPUs, *Computer Graphics International*, pp. 63–70 (2007).

[11] Kim, T., Thürey, N., James, D. and Gross, M.: Wavelet turbulence for fluid simulation, *ACM Trans. Graph.*, Vol. 27, No. 3, pp. 50:1–50:6 (online), DOI: 10.1145/1360612.1360649 (2008).

[12] Lefebvre, S., Hornus, S. and Neyret, F.: Octree Textures on the GPU, *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation* (Pharr, M., ed.), Addison Wesley, chapter 37, pp. 595–613 (2005).

[13] Macklin, M. and Müller, M.: Position based fluids, *ACM Trans. Graph.*, Vol. 32, No. 4, pp. 104:1–104:12 (online), DOI: 10.1145/2461912.2461984 (2013).

[14] Monaghan, J. J.: Simulating free surface flows with SPH, *J. Comput. Phys.*, Vol. 110, No. 2, pp. 399–406 (online), DOI: 10.1006/jcph.1994.1034 (1994).

[15] Müller, M., Charypar, D. and Gross, M.: Particle-based fluid simulation for interactive applications, *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp. 154–159 (online), available from ⟨http://dl.acm.org/citation.cfm?id=846276.846298⟩ (2003).

[16] Müller, M., Heidelberger, B., Hennix, M. and Ratcliff, J.: Position based dynamics, *J. Vis. Comun. Image Represent.*, Vol. 18, No. 2, pp. 109–118 (online), DOI: 10.1016/j.jvcir.2007.01.005 (2007).

[17] Orthmann, J. and Kolb, A.: Temporal Blending for Adaptive SPH, *Computer Graphics Forum*, Vol. 31, No. 8, pp. 2436–2449 (online), DOI: 10.1111/j.1467-8659.2012.03186.x (2012).

[18] Pelfrey, B. and House, D.: Adaptive neighbor pairing for smoothed particle hydrodynamics, *Proceedings of the 6th international conference on Advances in visual computing - Volume Part II*, ISVC'10, Berlin, Heidelberg, Springer-Verlag, pp. 192–201 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1940006.1940029⟩ (2010).

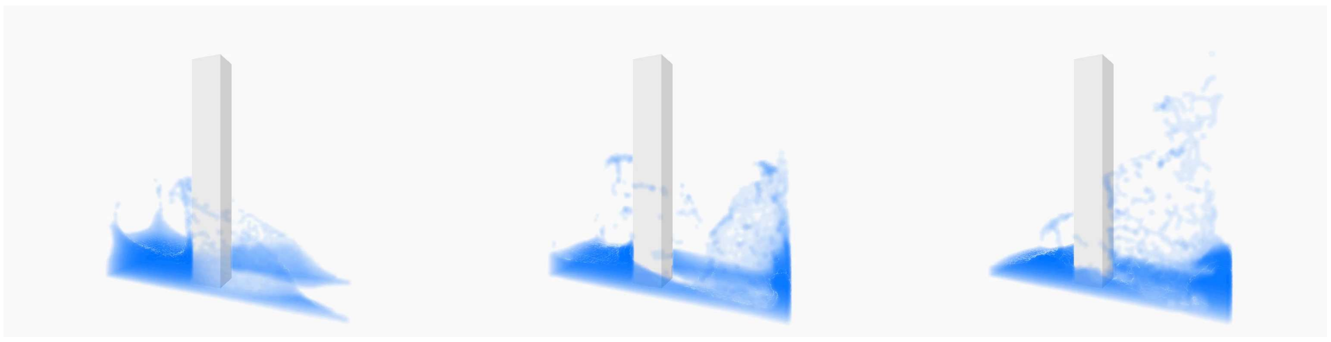
[19] Reeves, W. T.: Particle Systems – a Technique for Modeling a Class of Fuzzy Objects, *ACM Trans. Graph.*, Vol. 2, No. 2, pp. 91–108 (online), DOI: 10.1145/357318.357320 (1983).

[20] Solenthaler, B. and Pajarola, R.: Predictive-corrective incompressible SPH, *ACM Trans. Graph.*, Vol. 28, No. 3, pp. 40:1–40:6 (online), DOI: 10.1145/1531326.1531346 (2009).

[21] Solenthaler, B. and Gross, M.: Two-scale particle simulation, *ACM Trans. Graph.*, Vol. 30, No. 4, pp. 81:1–81:8 (online), DOI: 10.1145/2010324.1964976 (2011).

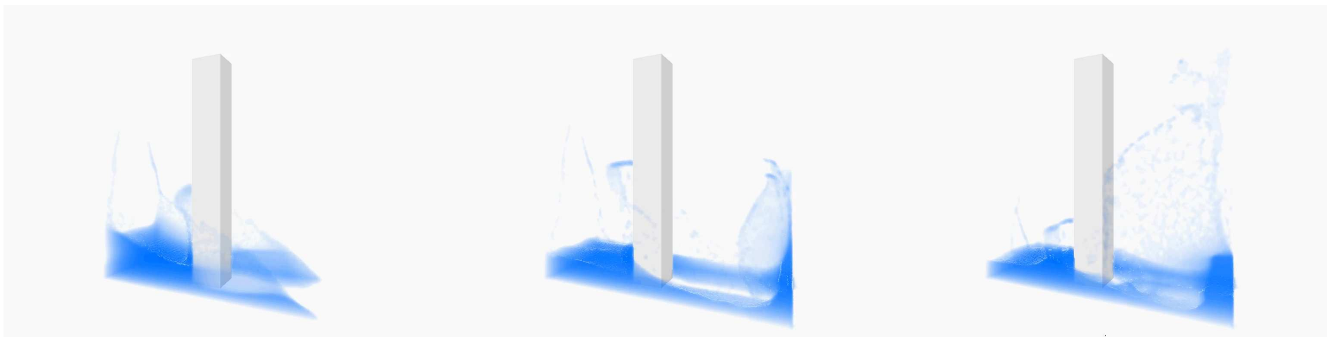
[22] van der Laan, W. J., Green, S. and Sainz, M.: Screen space fluid rendering with curvature flow, *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, New York, NY, USA, ACM, pp. 91–98 (online), DOI: 10.1145/1507149.1507164 (2009).

[23] Zhang, Y., Solenthaler, B. and Pajarola, R.: Adaptive Sampling and Rendering of Fluids on the GPU, *Proceedings Symposium on Point-Based Graphics*, pp. 137–146 (online), available from ⟨http://dx.doi.org/10.5167/uzh-9735⟩ (2008).



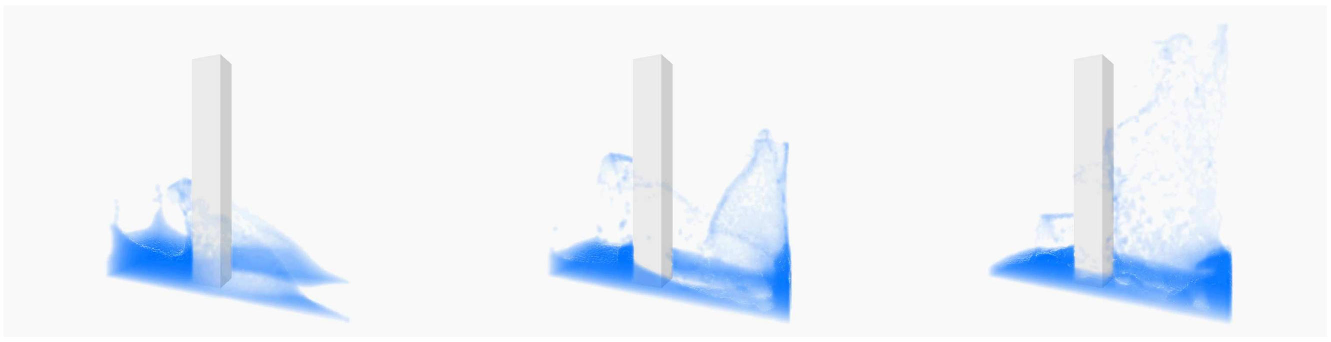
t = 500 ms　　*t* = 750 ms　　*t* = 1000 ms

Low-resolution simulation



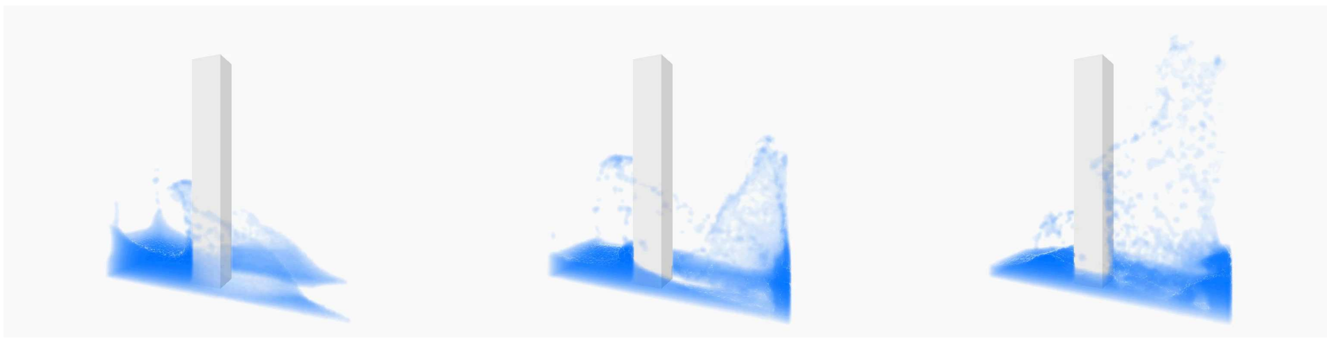
t = 500 ms　　*t* = 750 ms　　*t* = 1000 ms

High-resolution simulation



t = 500 ms　　*t* = 750 ms　　*t* = 1000 ms

Multi-resolution simulation (no merging)



t = 500 ms　　*t* = 750 ms　　*t* = 1000 ms

Multi-resolution simulation

Fig. 5: Visual result for all scenarios