Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2016)
Ladislav Kavan and Chris Wojtan (Editors)

Constrained Neighbor Lists for SPH-based Fluid Simulations

R. Winchenbach, H. Hochstetter and A. Kolb

Computer Graphics and Multimedia Systems Group, University of Siegen, Germany

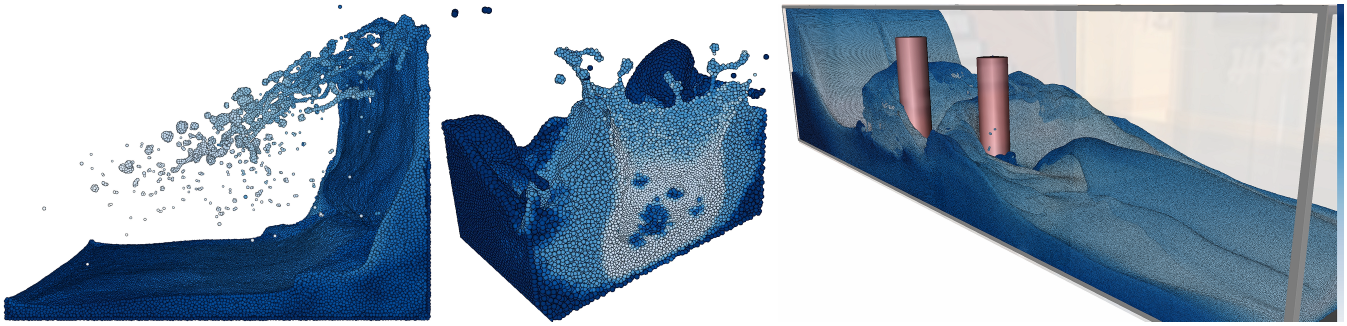

Figure 1: *Our improved neighbor algorithm can handle real time simulations of over 500K particles (left image, velocity color coded), multiple fluid resolutions at once (middle image, support radius color coded), and large scale simulations with over 35 million particles (right image, velocity color coded).*

Abstract

In this paper we present a new approach to create neighbor lists with strict memory bounds for incompressible Smoothed Particle Hydrodynamics (SPH) simulations. Our proposed approach is based on a novel efficient predictive-corrective algorithm that locally adjusts particle support radii in order to yield neighborhoods of a user-defined maximum size. Due to the improved estimation of the initial support radius, our algorithm is able to efficiently calculate neighborhoods in a single iteration in almost any situation. We compare our neighbor list algorithm to previous approaches and show that our proposed approach can handle larger particle numbers on a single GPU due to its strict guarantees and is able to simulate more particles in real time due to its benefits in regard to performance. Additionally we demonstrate the versatility and stability of our approach in several different scenarios, for example multi-scale simulations and with different kernel functions.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.6.8 [Simulation and Modeling]: Types of Simulation—Parallel

1. Introduction

The *Smoothed Particle Hydrodynamics (SPH)* method plays an important role in scientific computing and computer animation. Due to it's nature as a Lagrangian simulation it offers high spatial flexibility and support the simulation of incompressible fluids with free surfaces and various physical properties. In SPH fluids are described by unstructured particle data and local fluid quantities are interpolated from a set of particles within a compact support radius.

As these particle pairings need to be checked for every interaction it can be beneficial to store them in a neighbor list. Creating these neighbor lists has traditionally been very expensive on GPUs due to unbounded memory consumption and irregular access patterns [IABT11]. But they can still be used on GPUs if they are reused often in an iterative pressure solver [GEF15].

In this paper we introduce an efficient and versatile neighbor list method for incompressible SPH fluids simulations on GPUs with strict memory bounds and improved access patterns providing benefits in performance in all situations. To achieve this we propose a new formulation to locally adjust the particle support radius in every time step instead of using a fixed support radius. In order to guarantee our strict memory bounds we propose a predictive-corrective algorithm that correctly reduces the support radius of particles that violate the given bounds until they are correctly limited. Finally we propose a new structure to store the neighbor list in that improves access patterns and speeds up the overall simulation.

DOI: 10.2312/sca.20161222

Using our proposed algorithms we are able to simulate larger particle sets on a GPU and perform faster calculations when comparing it to previous approaches. Additionally we show how our method is able to handle multiple fluid resolutions and dynamic rigid boundaries.

2. Foundations and related work

Since the introduction by Gingold, Monaghan [GM77] and Lucy [Luc77] in the field of astrophysics, SPH has spread into many areas of research including, our area of interest, computer graphics [MCG03]. First designed for the simulation of compressible fluids, SPH has since been extended to support incompressible fluids [MM13, ICS*14, BK15], strong surface-tension effects [AAT13], two-way-interactions with rigid bodies [AIA*12] and many more effects. We refer the reader to the survey paper by Ihmsen et al. for a general overview [IOS*14].

In SPH, fluid quantities are evaluated by interpolating information of neighboring particles. The interpolant of a quantity A of particle i at its position $\mathbf{x}_i$ depends on the position $\mathbf{x}_j$, mass m_j, density ρ_j and quantity A_j of the neighboring particles j and is commonly written as

$$A(\mathbf{x}_i) = \sum_j A_j \frac{m_j}{\rho_j} W(x_{ij}, H), \tag{1}$$

where $x_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$ represents the distance between particles i and j [Mon05]. W is a kernel function that weights particle quantities based on x_{ij} and the support radius H. Interactions only take place if x_{ij} is shorter than the support radius H leading to a compact support radius. In practice, the dynamic particle volume $\frac{m_i}{\rho_i} = \widetilde{V}_i$ can be replaced by $\widetilde{V}_i = \frac{1}{\delta_i}$, where $\delta_i = \sum_j W(x_{ij}, H)$ is the particle density, in order to correctly handle density contrasts [SP08]. To stably handle free surfaces, Orthmann et al. presented another derivation based on the particle number density $n_i = \sum_j V_j W(x_{ij}, H)$ as $\widetilde{V}_i = \frac{V_i}{n_i}$, where $V_i = \frac{m_i}{\rho_{i,0}}$ is the particle's rest volume and $\rho_{i,0}$ its rest density [OHB*13].

The support radius either is uniform for all particles [IABT11, AAT13, AIA*12] or gets locally and dynamically adjusted for each particle to increase the simulation stability [Mon05, DA12] or in order to reduce the simulation time by only simulating at full particle resolution in areas of interest [SG11, OK12]. A common approach to calculate local support radii is given as

$$H_i = s_h \eta \left(\frac{m_i}{\rho_i} \right)^{\frac{1}{3}}, \tag{2}$$

where η is a configuration parameter set ideally somewhere between 1.2 and 1.3 [Mon05]. Note, we directly adjust the support radius in Eq. 2 instead of the smoothing length h_i [Mon05]. To be more consistent, we will only use the support radius throughout the text which is related to the smoothing length by the constant smoothing scale $s_h = \frac{H}{h}$ and depends on the shape of the kernel function [DA12]. In order to conserve quantities and momentum, particle interactions have to be symmetric. Thus, varying support radii are usually symmetrized as $H = \frac{H_i + H_j}{2}$ [Mon92, OK12]. Locally and dynamically adapting the support radius also introduces additional gradient H terms in each derivative. These additional terms, however, have neglectable effect [HK89], thus, they are usually omitted [HK89, OK12]. Although locally adapting the support radius is common practice, so far, no approach aimed at using it to strictly limit the memory consumption of simulations.

Calculating and accessing particle neighborhoods are core problems of every SPH framework. To that end, the simulation domain is usually subdivided by uniform grids [HKK07, GS10, Gre10, OK12] often in combination with spatial hashing [IABT11] or by hierarchical data structures [HK89, Gon15] into which particles are sorted. These data structures allow particles to be accessed based on their physical location. Often cells are ordered by space filling curves and to increase cache efficiency particles are sorted accordingly so that particles that are close neighbors in memory are also close neighbors in simulation space [GS10, IABT11, DCGGM11]. During simulations each particle has to traverse all possible neighbors in this data structure. Using hierarchical data structures is rather costly and thus usually only applied for compressible simulations with strongly varying support radii and gravitational codes [HK89]. For incompressible flows, uniform cells are commonly used with cell sizes of H^3. Then for each particle 27 cells have to be searched for neighbors. However, still about 87% of these potentially neighboring particles lie outside the particle's support radius and thus don't interact [DA12].

To prevent these spurious particle pairings, neighbor lists can be calculated which explicitly store all pairs of interacting particles [Ver67, DCGGM11]. Especially in incompressible simulations using iterative solvers [ICS*14, BK15], many particle interpolations have to be performed in each time step. Computing a neighbor list only once per time step instead of calculating all possible interactions for every interpolation strongly improves performance [IABT11, GEF15]. Neighbor lists can either be processed in two passes, a first pass to calculate the number of neighbors to allocate enough memory and a second pass to actually find the neighbors [VBC08], or by pre-allocating a fixed array with a maximum number of neighbors per particle [DCGGM11].

The simulation of SPH-based fluids can be efficiently carried out on GPUs using regular grids to subdivide the simulation domain [HKK07, Gre10, GS10, GEF15]. As hierarchical data structures and hashing involve irregular access patterns and cause thread divergence, they are usually avoided. Due to the restricted amount of memory, particle neighborhoods are often accessed by traversing cells for each interpolation [HKK07, Gre10, GS10]. Explicitly storing neighbor lists on the GPU [OK12, GEF15] can get very memory-consuming and thus strongly limits the number of particles if the neighborhood size is unbounded. So far, no approach to restrict the size of neighborhoods has been presented.

3. Variable support SPH

There is an ideal number of neighbors N_H inherent to every kernel function which depends on the shape of the kernel. For the cubic spline kernel the ideal number of neighbors is given as $N_H = 50$ [DA12]. For our improved neighbor list algorithm we propose a formulation of the support distance that is derived based on the ideal number of neighbors.

The support distance H_i and the number of neighbors N_H are

closely related as within a sphere of radius H_i only a certain number of particles with their respective volumes $\widetilde{V}_i$ can be found. This relation can be described as $N_H = \frac{\frac{4}{3}\pi H_i^3}{\widetilde{V}_i}$. Solving for H_i then yields our proposed formulation to locally adjust the support radius as

$$H_i = \widetilde{V}_i^{\frac{1}{3}} \underbrace{\left(\frac{N_H}{\frac{4}{3}\pi}\right)^{\frac{1}{3}}}_{s_h \eta}. \tag{3}$$

That way, we determine η depending on the properties of the kernel rather than by manually tweaking.

Our formulation for adapting the support radius is similar to the commonly used one, see Eq. 2. However, we use the adaptive particle volume of Solenthaler [SP08]

$$\widetilde{V}_i = \frac{1}{\sum_j W(x_{ij}, H)} \tag{4}$$

instead of $\frac{m_i}{\rho_i}$. In contrast to our general volume estimate, according to [OHB*13], we chose this formulation for the support distance as we are interested in the actual spatial distribution of the particles and not a corrected distribution that takes into account the particle volume. If the particle volume is taken into account, our estimate cannot adequately handle boundaries between particle resolutions as in those cases the adaptive volume would not significantly change due to its corrective effects. Equation 4 does not correct the summation for the different particle volumes and causes the desired change in the adaptive volume. For simulations of uniform particle rest volumes both formulations lead to the exact same result.

The proposed formulation locally adapts in two ways to the current simulation. On the one hand for surface particles without a full neighborhood our formulation increases the support distance and helps fill up the surface particle's neighborhood which can increase the SPH interpolation at the surface. On the other hand, for spatially compressed particles, we reduce the support distance and hence reduce the number of neighbors to meet the desired maximum.

In the first time step of a simulation, we can only assume an initial distribution where $\widetilde{V}_i = V_i$ as we cannot calculate the actual estimate without having a support distance. As we can assume temporal coherence of the particle neighborhood, we later use the support radius of the previous iteration to give better approximations for $\widetilde{V}_i$. To facilitate this we apply a linear interpolation of the newly predicted support radius H_i using Eq. 3 and the support radius of the previous timestep H_i^l as

$$H_i^{l+1} = H_i^l + \omega \left[H_i - H_i^l\right], \tag{5}$$

where ω is the weight which we usually set as $\omega = 0.5$. By exploiting temporal coherence, we often find an estimate of the support radius that directly yields the desired neighborhood size.

4. Constrained neighborhoods

Our proposed constrained neighborhood algorithm aims at efficiently calculating particle neighborhoods of a fixed, user-defined size. Using a fixed size allows giving strict bounds on the memory consumption and also allows us to calculate the neighbor list in a single pass over the underlying particle access data structure in most situations. Our algorithm works in an iterative way similar to a prediction-correction method using a soft start by initializing the support radius according to Sec. 3 and then predicting a neighborhood using this support distance and correcting potential errors. Note that the underlying data structure of our neighbor lists is an array. We only use the term list to be consistent with literature.

While our initial support radius gives good estimates, it is not able to guarantee neighbor limits as we always get a distribution of neighborhood sizes above and below the desired number N_H. Although our algorithm is able to enforce a strict limit of N_H neighbors, in some scenarios it can be beneficial to allow each particle to store N_{add} neighbors, so that in total each particle can have $N = N_H + N_{\text{add}}$ neighbors.

Fig. 2 shows a flow diagram of the proposed algorithm. In the iterative process we first try to create a neighbor list of fixed size (see Sec. 4.1). While adding neighbors to the list, we store the support radii of the farthest particles, i.e. the fringe, of the neighborhood in a *fringe buffer* of limited size and if necessary, additional particles are stored in an *overflow buffer* (see Sec. 4.2). If too many neighbors are found, the support radius is reduced using the fringe buffer. As neighboring support radii may also have been reduced, we try to replace corresponding neighbors with neighbors from the overflow list (see Sec. 4.3). Only if we run out of space in the overflow list, or cannot reduce the support radius sufficiently in a single iteration due to resource constraints, we have to do another iteration over the particle access data structure. Due to the soft start and the overflow buffer, in most simulation time steps the neighborhood can be calculated in a single iteration consisting of a single pass over the underlying data structure.

4.1. Initial neighborhood list

First, we calculate a neighbor list of at most N elements for each particle i using a single pass over the underlying data structure. The fixed space of the neighbor list, however, may not be large enough to directly contain all neighbors inside the support radius H_i.

In case, no particle finds more than N neighbors, we are done and the predicted support radius of Sec. 3 is valid for all particles. This corresponds to the green control flow in Fig. 2. In case any particle finds more neighbors, the respective support radius has to be decreased in order to reduce the number of neighbors and fit the strict memory bounds. In the worst case, additional iterations of the neighbor search have to be run.

4.2. Overflow lists and tracking the neighborhood fringe

In order to prevent additional iterations of the neighbor search, we keep track of neighbors that initially do not fit into the neighbor list. Therefore, we use an *overflow list* that is subdivided into fixed-size partitions. In this list we can store a limited number of inidices of additional neighbors that did not fit into the normal list without correction. For each particle we can only grab one partition and, in order to keep memory consumption low, the total number of partitions is limited.

During the construction of the initial neighbor list, we already

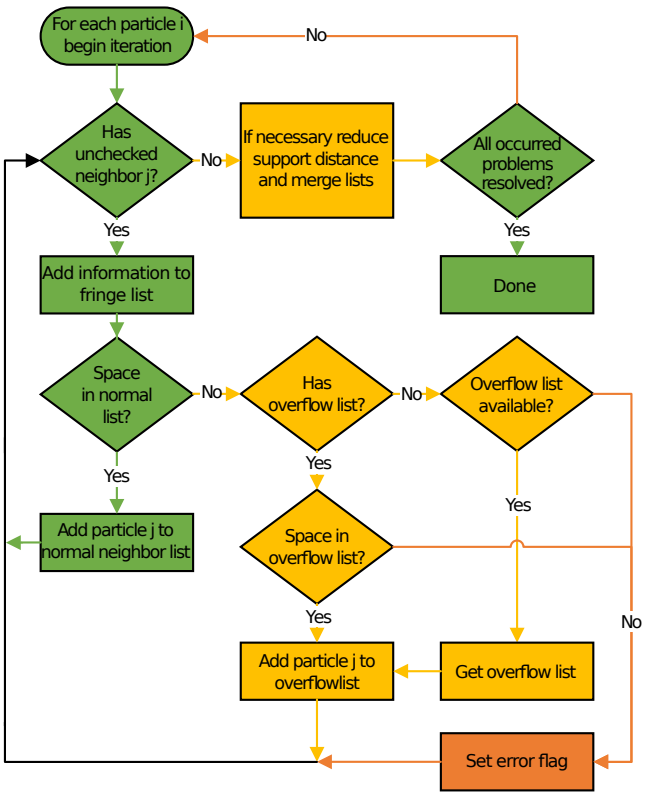

Figure 2: *Flow chart of our proposed neighbor algorithm. Green elements describe the path of the method that is always done. Yellow elements describe our added functions to handle the overflow and to correct the support distance. If the error flag is set (red box), our algorithm needs another iteration as it cannot accurately handle all errors in this iteration.*

keep track of the outermost neighbor particles j for each particle i in the fringe of the neighborhood, i.e. the farthest particles from i. To that end, we propose to use a *fringe buffer* that stores the largest possible support radii $H_i = 2x_{ij} - H_j$ for particle i so that the distance x_{ij} to the neighboring particle j is just inside the symmetrized support $\frac{H_i+H_j}{2} = x_{ij}$. For each particle the fringe buffer holds the same number of elements as a partition of the overflow list, however, the fringe buffer is stored in shared memory in order to allow for an efficient sorting of its elements. We only allow to store unique values in the fringe buffer and only replace the current minimum value if a value is found that is larger than the current minimum. Although it only occurs very rarely, if two or more elements had exactly the same value, the algorithm might not terminate otherwise. We split the storage of distances and indices as we need to sort the distances and replace values within the fringe buffer but only add values to the overflow list, see Fig. 2.

If a particle i has found M neighbors with $M > N$, we have to correct the support radius so that only N particles remain inside H_i. In order to find the proper support radius, we use the fringe buffer and sort it in descending order by distance. The $(M-N)$-th entry in the fringe buffer is then used to directly set the support radius of particle i as this is the support radius that particle i needs to have in order to just contain the targeted neighbor number. Note that as we only store unique values in the fringe buffer, it may rarely occur that the support radius is slightly over-corrected.

4.3. Merging of initial neighborhood and overflow list

If a particle has used an overflow partition, its support radius has been reduced in the previous step. As neighboring particles are likely to have found too many neighbors, too, these possibly can be removed from the neighborhood as well. Additionally, we want to merge the entries in the overflow list with the normal list.

In order to remove these particles, we simply iterate over the neighbor list to find particles that are now outside the support radius. These are replaced by particles from the overflow list that are within the support distance. That way, we effectively merge the initial neighbor list with the overflow list. Only if particles remain inside the support radius that do not fit into the neighbor list, we have to run another iteration to further correct the support radius.

To keep this process efficient we only apply it to particles with an overflow list and not for every particle in the simulation. Although this causes some spurious particle interactions, the impact on the simulation performance can be neglected due to the very limited number of spurious interactions. As we use symmetrized support radii in the SPH integration, particle interactions always remain symmetric so that conservation of momentum is guaranteed.

4.4. Discussion

A basic approach to strictly limit the size of the neighborhood is to use a constant support radius but only search up to N neighbors for each particle and then stop. This, however, causes simulations to get highly unstable because, on the one hand, the particle neighborhood is no longer symmetric and, on the other hand, the kernel's normalization property is violated if particles are removed from the neighborhood without properly adjusting the support radius.

In contrast, our proposed algorithm works by first predicting and then iteratively correcting the support radius in order to reduce the number of neighbors. As only the farthest particles of the neighborhood get removed, no instabilities due to asymmetric interactions occur and due to the fact that the support radius is properly adjusted, the normalization property of the kernel function is still satisfied. The support radius is reduced iteratively using a value from the fringe buffer. As the fringe buffer only contains unique values that are smaller than the current support radius, the support radius can only be reduced by our algorithm. Hence, the size of the neighborhood will also be reduced in each iteration and the algorithm terminates successfully.

5. Implementation details

5.1. Overflow lists

Although in most simulation scenarios only few particles find more neighbors than the user-defined limit, this would always cause a second iteration of the neighbor search. In order to prevent additional iterations, we store the overflow of the neighborhood lists in

an additional buffer (see Sec. 4.2) and later try to merge entries with the neighbor list. We set the size of this buffer so that each particle can have one additional neighbor. Each partition of the buffer has 12 elements which corresponds to the size of the fringe buffer that we only store in shared memory. We use a single counter in global memory that points to the first free partition in the buffer. If a particle needs an overflow partition, we use the current counter value as start address and increment the counter using an atomic-add operation. If more overflow lists than available are requested, an error flag is set and we have to start another iteration.

5.2. Constraining the initial support radius

In our implementation, we use a cell based particle access data structure with uniform cell sizes. The cell size limits the support radius that is allowed for a particle without clipping the neighborhood. Initially, the cell length is set to fit a particle with no compression (see Eq. 3) and we later limit support radii to this length.

5.3. Coalesced neighbor lists

One technical improvement, we propose, is the use of a coalesced neighbor list. In traditional neighbor lists [OK12], every particle stores its neighbors contiguously in memory. This is a very straight forward implementation, however, on GPUs this approach does not perform well due to non-coalesced loads and small cache sizes per particle. In order to resolve this issue, we propose an optimized data structure in which the classical neighbor list is stored in transposed order. So we contiguously store the first neighbor of every particle and then the second neighbor of every particle and so on. This optimization allows us to access the neighbor list in fully coalescing loads without cache problems.

Note, this structure is only realistically possible if the length of the neighbor list for each particle is strictly limited. Otherwise, the size of the data structure would have to be adjusted to the particle with the most neighbors which would be highly restrictive to simulating large particle numbers.

6. Results and discussion

In order to evaluate the runtime performance and memory consumption and in order to show the versatility and stability of our approach, we ran several test cases. To enforce fluid incompressibility, we used IISPH [ICS*14] and DFSPH [BK15] resulting in an average compression rate of 0.5%. Dynamic rigid objects were realized using particles [AIA*12]. For static boundaries we used distance fields [HKK07]. We used the model of Akinci [AAT13] to simulate surface tension effects. Simulations were run on an Intel i7-5930K with 16GB of host RAM and an Nvidia Geforce Titan X with 12GB of device RAM. Fluid surfaces were rendered using screen-space curvature smoothing [vdLGS09].

First, we present the capabilities of our approach in real-time simulations. Then, we will give a comparison to previous neighbor algorithms in terms of memory consumption and computational speed and present simulations of very large scale. In order to demonstrate the stability of our method, we show complex scenarios of multi-scale simulations, rigid-fluid coupling and test the compatibility with different kernel functions. Finally in order to demonstrate the scaling and performance of our method we test our approach against previous approaches for a dambreak scenario using various resolutions.

Table 1: *Performance characteristics of the Bunny scene for different neighbor algorithms. 'Frametime' gives the run time of a simulation time step. 'Corrected' gives the number of particles for which our algorithm adjusted the support radius. 'Ratio' gives the time spent to calculate the neighbor list and sort the particles vs. the total time. 'Size' is the neighbor list size in device memory.*

Algorithm	Frametime (ms)	Corrected	Ratio	Size (MB)
Our approach $N = N_H = 50$	527	51923	9.07%	475
Constant H, coalesced	596.39	-	8.40%	932.61
Two-pass [OK12]	1323.39	-	3.33%	699.457
Cell iteration [Gre10]	765.83	-	1.2%	-

6.1. Real-time simulations

For real-time simulations, at least 30 frames have to be simulated per second. With our improved algorithm we were able to run a real-time simulation of a dam break scenario with over 500K particles using DFSPH and the poly6 kernel, see Fig. 1 (left). Using the cubic spline kernel, still 240K particles were simulated in real-time.

6.2. Scaling and comparison to previous work

We compared our new method to three previous approaches by simulating a bunny that was sampled with fluid particles and dropping it into a basin (see Fig. 3). The simulation ran with 2.4M particles for 15s simulated time using IISPH with cubic spline kernel. Table 1 gives the performance results averaged over all time steps.

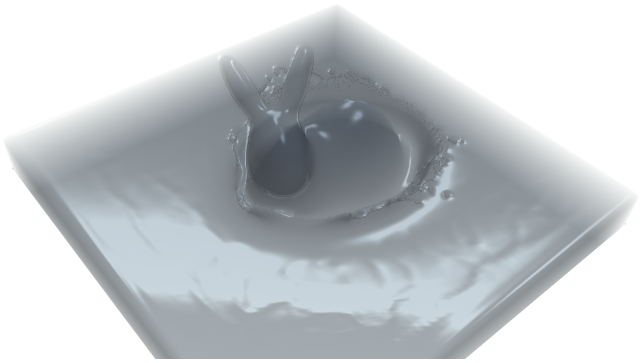

Figure 3: *A bunny is sampled with fluid particles and dropped into a basin. Surfaces are rendered using curvature smoothing.*

We ran our approach with $N = 50$ and outperformed all other approaches in simulation speed and neighbor list based approaches in memory consumption. The 'Two-pass' algorithm calculates the neighbor list in two passes over the particle access data structure and stores the exact number of neighbors for each particle contiguously in memory [OK12]. In comparison, our algorithm reduced

the memory consumption by 47%. Figure 4 shows the memory consumption of the neighborhood sizes resulting from using constant support radius and our adaptive support radius. Most particles in the fluid volume have more than N neighbors using constant support. Our method, however, is able to strictly enforce the desired number of neighbors. The row 'Cell iteration' corresponds to the approach in which the particles neighborhood is recomputed for every interpolation [Gre10]. It does not need any memory to store neighbor lists but took over 45% longer than our proposed approach to simulate. The row 'Constant H, coalesced' gives the results for a neighbor list approach with fixed size that uses our proposed coalesced data structure. By strictly limiting the neighborhood size, we were able to reduce the computation time significantly whilst consuming less memory than previous neighborlist based approaches.

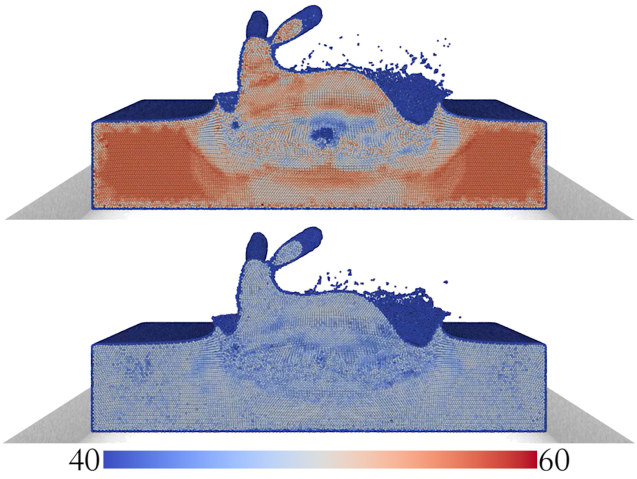

Figure 4: *Comparison of the Bunny scene using constant support radius and our method using $N = 50$. The number of neighbors is color coded.*

In order to evaluate the scaling of our method in comparison to previous works, we simulated 15 seconds of a dambreak scenario (see Fig. 10) using various particle resolutions yielding simulations of 30K to 4M particles. We compared our algorithm using different neighborhood sizes $N = 50, 55, 60$ with the naive neighbor list approach (Two-pass [OK12]), a cell based approach (Cell iteration), and an approach using our proposed coalesced data structure without constraints (Coalesced). Fig. 5 shows the average overall frametime of the different approaches for different particle resolutions. Our method was able to handle small and large simulations very well and outperformed the previous approaches for all particle resolutions in terms of the overall frametime. This is due to the fact that our method creates a lower average number of neighbors than previous approaches, as shown in Fig. 6, so that the costly time integration of the SPH equations is sped up considerably whilst using an efficient access structure. On average our approach found about 10% less neighbors than methods using a constant support radius for the cubic spline kernel, when using strict constraints of N=50 neighbors, without negatively influencing the simulation behavior. The smaller N is chosen, the smaller the average number of neighbors gets. Fig. 7 shows the ratio of the SPH integration and

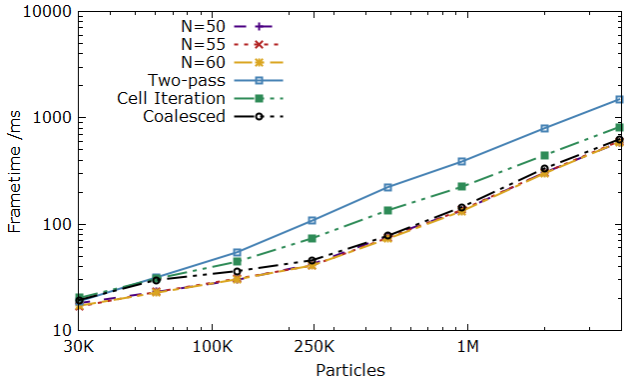

Figure 5: *Average time to simulate one frame (y-axis) for different particle resolutions (x-axis) of our neighbor algorithm (N=50, 55, 60) and previous works.*

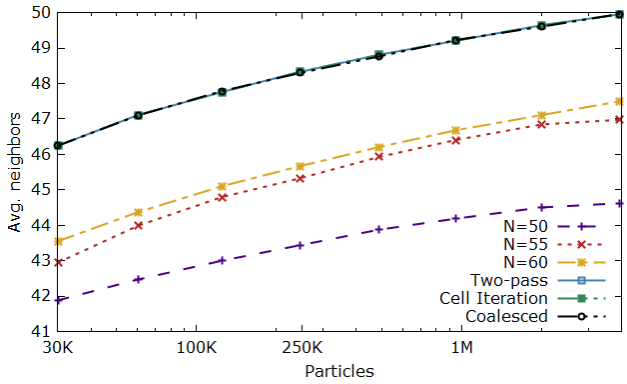

Figure 6: *Average number of neighbors per particle (y-axis) for different particle resolutions (x-axis) of our neighbor algorithm (N=50, 55, 60) and previous works. Note that Two-pass, Cell iteration and Coalesced do not adjust support radii and thus all yield the same number of neighbors.*

the overall frametime, the remaining time is spent for the neighborhood algorithm. With our approach relatively more time is spent for finding neighboring particles than for the integration. Although our method introduces a larger computational overhead over previous approaches, it yields smaller neighborhood sizes and hence considerably reduces the time spent for the time integration and is able to speed up the overall simulation time.

6.3. Large scale simulations

One of the main benefits of the proposed approach is that more particles than with previous neighbor list approaches fit into memory. A large scale scenario with up to 35M particles in which a stream of water is perturbed by two pillars could be simulated, see Fig. 1 (right). The simulation took 3.5 s per simulation step and the neighbor list consumed a total of 2240MB with a strict limit of $N = N_H = 15$ neighbors using IISPH and the poly6 kernel. The only

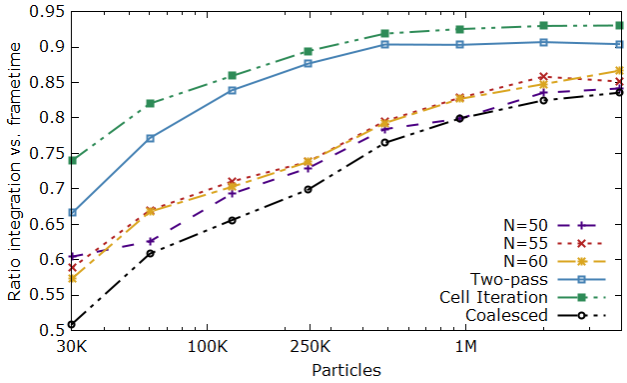

Figure 7: *Ratio of the time spent in the integration part of the simulation step over the overall simulation frametime (y-axis) for different particle resolutions (x-axis).*

other method able to simulate 35M particles was the cell iteration. However, it took about 6.8s to simulate a time step.

6.4. Stability and versatility of our approach

Without additional adjustments, our method was able to stably simulate multi-scale scenarios with volume ratios of 1:8 using $N_{\text{add}} = 25$ additional neighbors (see Fig. 8). We were also able to handle volume differences with a ratio of 1:2 without additional neighbors required. We also tried to run simulations in which we

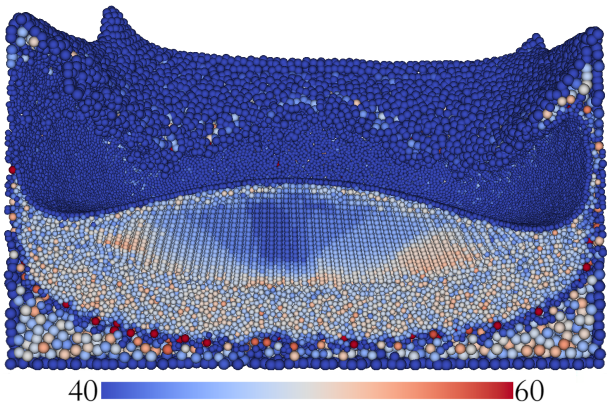

Figure 8: *A spherical drop of particles with $r = 0.5m$ drips into a fluid volume with $r = 1.0m$. We set $N = N_H + 25$ to handle the boundary between particle resolutions. For an unbounded neighbor list we set $N = N_H + 150$. The number of neighbors is color coded.*

stopped adding neighbors to the neighbor list after $N = N_H$ neighbors had been found. This however caused severe instabilities due to asymmetric interactions. We thus omit to show these simulations.

Additionally, our method works with two-way rigid-fluid coupling (see Fig. 9) where the fluid particles are directly influenced by dynamic rigid particles.

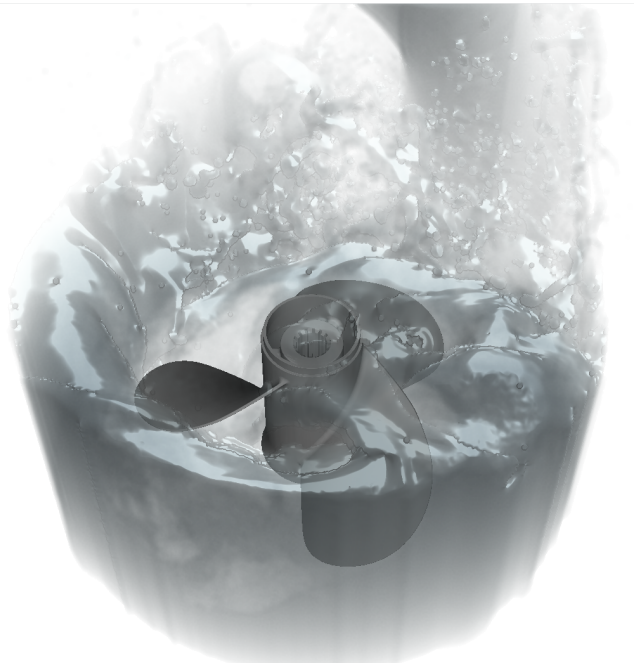

Figure 9: *A mixer is filled with 1.3M particles. The spinning rotor is simulated using dynamic rigid particles. The neighborhood size was set to $N = N_H$. Surfaces were rendered using curvature smoothing.*

During our experiments, we also evaluated the effect of different N_{add} for single-scale simulations using the cubic spline kernel. It showed that only in combination with multi-scale simulations additional neighbors were necessary to achieve stable simulations. Both for run time performance and memory consumption, $N_{\text{add}} = 0$ yielded the best results.

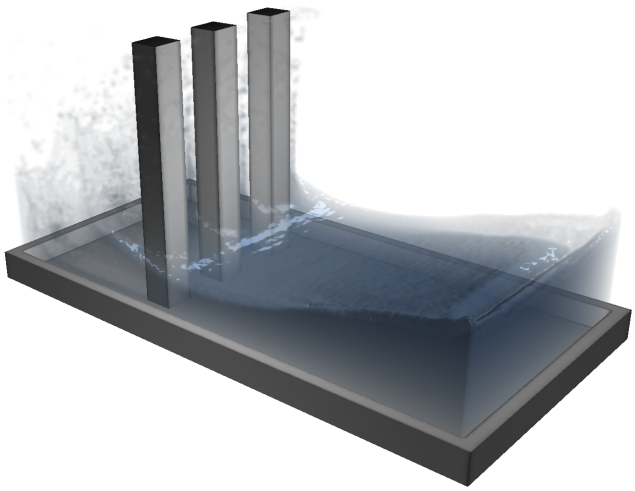

Figure 10: *Dam break scenario with 500K particles. Surfaces are rendered using curvature smoothing.*

Although for interactive simulations poly6 or cubic spline kernels are usually employed, more stable kernels are commonly used in scientific computing [DA12]. We were able to stably simulate the Dam Break scenario of Fig 10 using seven different kernel functions (poly6, cubic spline, quartic spline, quintic spline, Wendland2, Wendland4, Wendland6) and set $N = N_H$ to their respective ideal neighborhood size [DA12].

When using variable support radii, both the particle positions and support radii are time-dependent. Although, in general, this has an influence on the derivation of time-derivatives, we omitted to take derivatives of H_i into account as their effects have been shown to be neglibile [HK89] and we did not observe any issues concerning simulation stability by ignoring these terms.

7. Conclusions

In this paper, we presented a novel algorithm to efficiently calculate memory-constrained neighbor lists for SPH-based particle simulations. The approach works by iteratively adapting each particle's support radius so that a user-defined maximum number of neighbors is never exceeded. The algorithm works iteratively in a predictive-corrective way, where the proposed initial prediction is based on the current simulation state. In order to improve performance, an optimized data structure for neighbor lists has been proposed that allows for fully coalesced memory reads on GPUs.

Due to the restricted neighborhood size, both performance and memory consumption can be considerably improved compared to previous approaches. We are able to stably simulate incompressible fluids including two-way fluid-rigid coupling and multi-scale simulations. Due to our highly efficient method, over 500K particles could be simulated in real-time and, due to the strictly limited memory consumption, up to 35M particles could be simulated on a single consumer GPU.

References

[AAT13] AKINCI N., AKINCI G., TESCHNER M.: Versatile Surface Tension and Adhesion for SPH Fluids. *ACM Trans. Graph. 32*, 6 (2013), 182:1—-182:8. doi:10.1145/2508363.2508395. 2, 5

[AIA*12] AKINCI N., IHMSEN M., AKINCI G., SOLENTHALER B., TESCHNER M.: Versatile rigid-fluid coupling for incompressible SPH. *ACM Trans. Graph. 31*, 4 (2012), 1–8. doi:10.1145/2185520.2335413. 2, 5

[BK15] BENDER J., KOSCHIER D.: Divergence-Free Smoothed Particle Hydrodynamics. *Proc. 2015 ACM SIGGRAPH/Eurographics Symp. Comput. Animat.*, 1 (2015). doi:10.1145/2786784.2786796. 2, 5

[DA12] DEHNEN W., ALY H.: Improving convergence in smoothed particle hydrodynamics simulations without pairing instability. *Mon. Not. R. Astron. Soc. 425*, 2 (2012), 1068–1082. arXiv:1204.2471. 2, 8

[DCGGM11] DOMÍNGUEZ J. M., CRESPO A. J. C., GÓMEZ-GESTEIRA M., MARONGIU J. C.: Neighbour lists in smoothed particle hydrodynamics. *Int. J. Numer. Methods Fluids 67*, 12 (dec 2011), 2026–2042. arXiv:fld.2481. 2

[GEF15] GOSWAMI P., ELIASSON A., FRANZÉN P.: Implicit Incompressible SPH on the GPU. In *Work. Virtual Real. Interact. Phys. Simul. VRIPHYS* (2015). 1, 2

[GM77] GINGOLD R. A., MONAGHAN J. J.: Smoothed particle hydrodynamics-theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society 181* (1977), 375–389. 2

[Gon15] GONNET P.: Efficient and Scalable Algorithms for Smoothed Particle Hydrodynamics on Hybrid Shared/Distributed-Memory Architectures. *SIAM J. Sci. Comput. 37*, 1 (2015), C95–C121. arXiv:1404.2303. 2

[Gre10] GREEN S.: Particle Simulation using CUDA. *Cuda 4.0 Sdk*, May (2010). 2, 5, 6

[GS10] GOSWAMI P., SCHLEGEL P.: Interactive SPH simulation and rendering on the GPU. *Proc. 2010 ACM SIGGRAPH/Eurographics Symp. Comput. Animat.* (2010), 55–64. 2

[HK89] HERNQUIST L., KATZ N.: TREESPH - A unification of SPH with the hierarchical tree method. *Astrophys. J. Suppl. Ser. 70* (1989), 419. doi:10.1086/191344. 2, 8

[HKK07] HARADA T., KOSHIZUKA S., KAWAGUCHI Y.: Smoothed Particle Hydrodynamics on GPUs. *Computer Graphics International* (2007), 63–70. 2, 5

[IABT11] IHMSEN M., AKINCI N., BECKER M., TESCHNER M.: A parallel SPH implementation on multi-core CPUs. *Comput. Graph. Forum 30*, 1 (2011), 99–112. arXiv:1110.3711. 1, 2

[ICS*14] IHMSEN M., CORNELIS J., SOLENTHALER B., HORVATH C., TESCHNER M.: Implicit incompressible SPH. *IEEE Trans. Vis. Comput. Graph. 20*, 3 (2014), 426–435. doi:10.1109/TVCG.2013.105. 2, 5

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: SPH Fluids in Computer Graphics. *Eurographics STARS*, 2 (2014), 21–42. 2

[Luc77] LUCY L. B.: A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal 82* (1977), 1013. arXiv:9809069v1, doi:10.1086/112164. 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-Based Fluid Simulation for Interactive Applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 5 (2003), 154–159. 2

[MM13] MACKLIN M., MÜLLER M.: Position based fluids. *ACM Trans. Graph. 32*, 4 (2013), 1. doi:10.1145/2461912.2461984. 2

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics 30*, 1 (1992), 543–574. arXiv:arXiv:1007.1245v2, doi:10.1146/annurev.astro.30.1.543. 2

[Mon05] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Reports Prog. Phys. 68*, 8 (2005), 1–34. arXiv:0507472v1. 2

[OHB*13] ORTHMANN J., HOCHSTETTER H., BADER J., BAYRAKTAR S., KOLB A.: Consistent surface model for SPH-based fluid transport. *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '13* (2013), 95–103. 2, 3

[OK12] ORTHMANN J., KOLB A.: Temporal blending for adaptive SPH. *Computer Graphics Forum 31*, 8 (2012), 2436–2449. 2, 5, 6

[SG11] SOLENTHALER B., GROSS M.: Two-scale particle simulation. *ACM Trans. Graph. 30*, 4 (2011), 1. doi:10.1145/2010324.1964976. 2

[SP08] SOLENTHALER B., PAJAROLA R.: Density Contrast SPH Interfaces. *ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2008), 211–218. 2, 3

[VBC08] VICCIONE G., BOVOLIN V., CARRATELLI E. P.: Defining and optimizing algorithms for neighbouring particle identification in SPH fluid simulations. *International Journal for Numerical Methods in Fluids 58*, 6 (oct 2008), 625–638. arXiv:fld.1. 2

[vdLGS09] VAN DER LAAN W. J., GREEN S., SAINZ M.: Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 91–98. 5

[Ver67] VERLET L.: Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Physical Review 159*, 1 (1967), 98–103. 2