Fluid initialization and dynamic window for SPH simulation

Samuel Carensac[1], Nicolas Pronost[1] and Saida Bouakaz[1]

Abstract

Fluid simulation is an essential tool to produce realistic looking animations. In particular, Lagrangian simulations offer interactive computation times with an easy integration of the two-way interaction with rigid bodies. However, the interactivity is lost for larger scenes even if only the areas around the bodies have any visual interest. In this paper, we present a novel approach to quickly initialize additional fluid in a rest state in simulations with any 3D boundary shape and able to preserve any already existing fluid. Our approach only uses the density property of the particles to allow compatibility with any smoothed-particle hydrodynamics (SPH) simulation scheme and any boundaries model. This initialization method is fast enough to allow the initialization of new fluid volumes interactively while the simulation is running. We showcase our approach by proposing a method to create a dynamic simulation window, allowing the restriction of simulating the fluid only around moving objects. We propose multiple experiments to demonstrate the capabilities and performance of our approach.

Keywords

fluid simulation, boundary modelling, SPH implementation

1 Introduction

Nowadays, physics-based simulations are widely used to create animations. This is due to the increasing requirement to have more and more real life-like scenes where the virtual world behaves in a physically realistic way. One noticeable widespread type of physics simulations is fluid simulation and most animation software currently integrate at least a simple one. Through the years, multiples models have been proposed to simulate a fluid, from Eulerian approaches to pure Lagrangians approaches and hybrid approaches. Although, currently the hybrid PIC/FLIP (Particles In Cell/Fluid Implicit Particle) approaches are the most common in animation for its high graphical qualities, pure Lagrangian approaches are often used, but not limited, to produce animations at interactive computation times. Still, even when the computation time is not a limitation, fluid simulation can be limited by its requirement for large amount of memory. This problem is particularly noticeable for large scenes with moving objects where only the areas close to the objects have a real interest, leading to a large waste of memory space and computation time if the whole scene is simulated at all time to maintain the physics realism.

Solutions have been proposed to overcome this problem. One approach currently integrated in some animation software consists in only simulating with high quality the area of interest and using a lower resolution procedural animation for the remainder of the scene. However, it only considers a static area of interest and cannot handle the displacement of the area of interest during the animation to be able to follow the moving objects. A more complete solution has been proposed by Stomakhin and Selle [1]. Their solution handles any displacement of the simulated area on a procedurally generated background ocean. However, this solution can only be applied to hybrid fluid simulations. To our knowledge, no solution has been proposed for pure Lagrangian fluid simulations. One of the main conditions to create a dynamic simulation window is the ability to add fluid in the simulation when the window moves. Unlike for hybrid approaches, there is currently no existing model that can initialize additional fluid in a realistic state within a short computation time.

Our goal is therefore to propose a novel approach to initialize a fluid in a rest state for particle-based simulations for any 3D boundary shape and capable of integrating any fluid that already exists in the simulation area (section 3). We aim to propose a method that is fast enough to be used to initialize fluid while the simulation is running, while being completely independent of both the smoothed particles hydrodynamics (SPH) simulation scheme and the boundary model.

Our initialization method is then combined with a fast perturbation absorption system at boundaries to create a dynamic window for SPH fluid simulations (section 4). We show the versatility and low computation times of our initialization method through various experiments (section 5). We then present the capabilities of our perturbation absorption system and the application of our dynamic window to boat simulations (sections 5.6 and 5.7).

[1]Univ Lyon, UCBL, CNRS, INSA Lyon, LIRIS, UMR5205, F-69622 Villeurbanne, France

Corresponding author:

Nicolas Pronost, batiment Nautibus, 8 boulevard Niels Bohr, 69622 Villeurbanne, France.

Email: nicolas.pronost@univ-lyon1.fr

Finally, we discuss various points that could benefit from further studies and improvements (section 6).

2 Related works

2.1 Simulation window and boundaries

As it can be seen in the system proposed by Stomakhin and Selle[1], two main components are required for a dynamic window. The first component is a boundary model to handle the particle transfers between the inside and the outside of the high quality simulated area. This model has received quite a lot of interest in the context of SPH simulations for which various approaches have been proposed. To be able to simulate a high quality area nested within a larger lower quality area or a procedural model and to handle it is necessary to handle the transfer of information between the two areas. The most common approach is to replace the normal solid boundaries that block the particles by boundaries allowing the transfer of particles. They are called open boundaries and allow waves and flow of fluids to enter and exit the high quality simulated area giving the illusion that the high quality simulation extends further than the actual boundary. The main difficulty of such system is that the boundaries are often represented as solid static particles with constant mass making the transfer of particles though the boundary impossible. This problem may be solved by using cyclic or periodic boundaries[2]. In this approach, the flow of fluid at the paired boundaries must be identical which limits its usage as there is no real transfer of particles between the inside and the outside of the simulation space.

Currently, the most commonly used approach to generate true open boundaries is to replace the solid boundary by a buffer of manually placed particles as proposed by Lastiwka et al.[3]. Tafuni et al.[4] improved this method by using ghost particles to better compute the properties for the particles inside the buffer. They improved this approach several times[5,6] to allow for a single particle buffer to be both an inlet and outlet, which allows the particles to travel both ways at a boundary. Kassiotis et al.[7] proposed another approach where the mass of the nodes in a semi-analytical boundary model is manipulated in order to integrate the inflow and outflow of the fluid. This approach has been later improved by adding Riemann solvers to compute the properties of the particles when they are created at the boundary[8] and made more flexible in[9] by adding the possibility to specify a desired pressure at the boundary instead of the fluid velocity. Currently, open boundaries in Lagrangian simulations have mainly been used to couple the particle-based simulation with other models. Research works with that objective mainly use the buffer-based approach[10,11] or more rarely Kassiotis et al.'s approach[12]. All these works proposed open boundaries for a static area but do not explore the possibility of moving an area of interest in a larger simulated space which is our goal.

Few Eulerian-based works allow interactive use of a large scale volume of liquid. Among them, Chentanez and Müller[13] propose an hybrid approach combining particle, 3D grid and height field methods. Cells away from the liquid surface are grouped into larger cells on which simplified versions of the physical calculations are used. This technique saves computing time where the simulation does not have a significant visual impact. In the Lagrangian approach, in addition to influencing the accuracy of calculations, the particle size also affects the memory size needed to simulate a given volume. A simulation with larger particles limits the details obtained but makes it possible to simulate a larger volume of fluid without requiring prohibitive calculation times. This makes the use of dynamic particle sizes much more interesting than the use of dynamic time steps. Works using this approach showcase the ability of a fully dynamic particle size by merging and splitting particles depending on the precision required[14] or combine multiple simulations each with a static particle size[15]. However, this approach relies on a continuous transition between the particle sizes and the applicability of this approach to transitioning a significant amount of particles at a single simulation step, which would be necessary when moving a dynamic window, has not been explored.

The second component for the dynamic window proposed by Stomakhin and Selle[1] is a system to add fluid in the direction of motion and to remove fluid in the opposite direction. To our knowledge, no method exists to generate a distribution of pure Lagrangian particles similar to a fluid at rest, i.e. a fluid with no internal motion, that can be used to add and remove the fluid. When using Lagrangian simulations, most works start the simulation with particles arranged on a regular grid. This is not a possible solution when we desire to add a fluid that is initially at rest as such distribution generates large rearrangements of the particles and in particular near the borders. In his initial work, Monaghan[16] proposed to use a damping term to accelerate the rate at which the kinetic energy is expended. However, even with this improvement, waiting for the fluid to reach a rest state may still take a long simulation time.

2.2 Particle Packing

To reduce the computation time to reach a rest state, Colagrossi et al.[17] proposed a particle packing algorithm which displaces the particles depending on the gradient of the particles concentration instead of the pressure. Although their system is still an iterative process, less computations are required at each step resulting in faster computation times. Negi and Ramachandran[18] improved this approach by adding a repulsive force between the particles, making it more stable and compatible with 3D simulations (see Figure 2 and Algorithm 3 in[18]). Their results show that it improves the homogeneity of the density function in the fluid. However, this approach is significantly impacted by the quality of the solid particles distribution and potentially requires boundaries defined by multiples layers of solid particles. It makes the particle packing inapplicable for single layer boundary systems such as Akinci et al.[19] and for particle-less boundaries such as Bender et al.[20]. Moreover, the use of solid particles at the free surface can be problematic as, depending on the simulated scene, sampling a boundary above the free surface can be difficult, for instance in a wavy scene with floating objects.

2.3 Pre-simulated volume

Due to the objective of doing repeated initialization for an optimization, Hall et al.[21] noted that, even if the particle

packing approach is faster than a damped simulation (i.e. simulating a non negligible number of steps in order for the fluid to stabilize and remove the local high and low pressure regions), its iterative nature is still too slow for its application. As such, instead of starting with the usual cartesian or triangular initial particle distribution, they decided to initialize the fluid with particles already in a rest state distribution (see Figure 10 in[21]). To obtain the rest state distribution of the particles a volume of fluid enclosing all the volumes of fluid they aim to initialize during their optimizations is simply pre-simulated and stored. The method requires a dependency between the values of the boundary particles and the fluid distribution that may lead to uncertain solid definition as, depending on the fluid particles distribution, the properties of the solid particles will change. Also, this solution is limited to boundary models that use solid particles. The approach is unlikely to be applicable to boundary models requiring special computations for the boundary particles properties such as Akinci et al.[19]'s model. Also, this approach removes all fluid particles within one kernel radius of the boundary, creating large gaps between the fluid and the solid. These gaps may be reduced by adapting the solid particles properties. However, this will lead to the loss of information about the geometry if the particles are not small enough. Finally, as the method needs to modify the mass of the boundaries particles to fit the loaded distribution, it is inapplicable for any simulation already containing some fluid.

In this paper we propose an algorithm to initialize a distribution of particles corresponding to a fluid at rest. It is fast to compute and compatible with complex and versatile 3D boundary models and SPH formulations while being completely independent of the chosen SPH scheme and boundary model. We then use our algorithm in combination with a perturbation absorption system to design a novel dynamic simulation window process for particle based simulations. It allows us to interactively move a high resolution simulated area within a virtually infinite simulation space in real-time.

Our framework is capable of simulating moving objects in interaction with a virtually infinite 3D volume of fluid. It would be used in high-performance interactive applications like video games, but also visual effect software to simulate for instance floating or immersed objects, or characters walking in a fluid that a camera will follow.

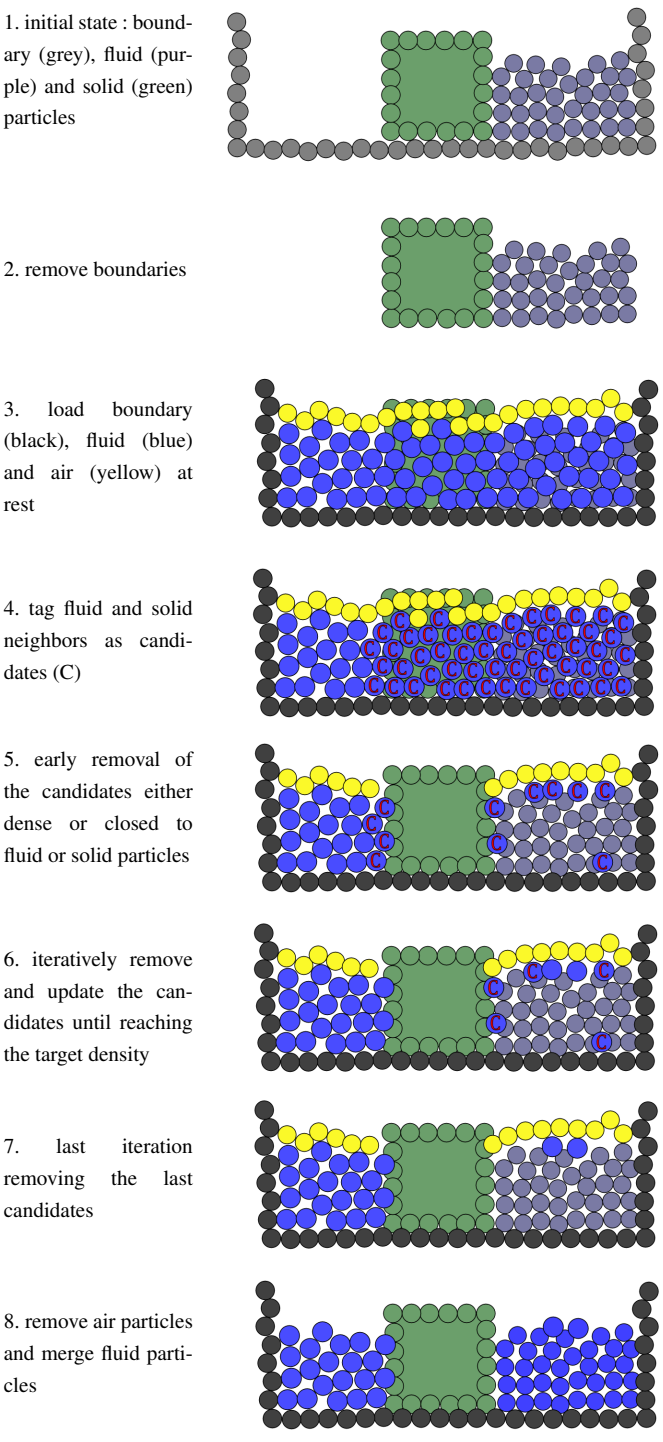

Figure 1. Overview of the rest fluid initialization algorithm.

3 Fluid initialization

Our algorithm follows Hall et al.[21]'s idea of pre-computing and saving a larger simulation space with the fluid at rest and then loading only the particles inside the simulation area at execution time. However, instead of not loading any particle within the smoothing length of the boundaries, we use a condition based on the density in order to select the particles that will be loaded (see section 3.1 and section 3.2). It allows us to remove the modification of the boundary particle mass. On top of removing a drawback, this modification should make the method easily applicable for any boundary model as our only requirement is the ability to compute the density of the fluid particles. We end our loading process with a few simulation steps to obtain a perfect stabilization of the fluid (see section 3.3). Although our selection process uses an iterative algorithm, it is only a selection process and no particles are moved in that process. As such, the number of iterations is much lower than in a damped simulation or in a particle packing process. Before explaining our selection process we must note one additional difference with Hall et al.[21]'s approach. Since we compute the particles density, we must load additional particles around the volume of fluid we are trying to initialize. These additional *air* particles are necessary to have a complete neighborhood when computing the density of a fluid particle. We can load the air particles by loading every particle that is inside the simulation boundaries

and in the neighborhood of one of the fluid particles without being a fluid particle itself. The air particles will be removed from the simulation once the selection process is finished, and therefore never rendered. Since we only compute the density of a limited amount of particles, it is only necessary to load the air particles that are around the specific areas of interest.

3.1 Fluid particle selection

The first step in our algorithm (see Figure 1, Algorithm 1 and Appendix B) is to select the fluid particles that have to be loaded. We start by loading all particles that are inside the required simulation space. We then eliminate the particles that are extremely close to the boundary in order to accelerate the following processing. This step is required since, if there is a particle that is too close to the solid boundaries when running the simulation, it will produce a high particle velocity that can either produce an artifact or even tunnel the particle through the boundary. Any particle that has an overlap of at least 50% with a boundary particle is removed, and in practice it means removing any particle that has a distance to the boundary smaller than $D = 0.5 \times particleRadius$, making it compatible with particle-less boundary models and a bit faster to compute. We must precise that in our experiments the boundary and other solid objects are sampled by placing the solid particles on the surface of the mesh. If the particles for a solid object were placed adjacent to the inside of the mesh then the distance to the boundary would be $D = 0$. A distance $D = 0$ could be used, respectively $D = -0.5 \times particleRadius$ if the solid objects are sampled with the solid particles inside of the boundaries, meaning only removing the particles that are strictly outside the simulated area. However, we have found in our experiments that no particle with a 50% overlap remains once the particles selection step is finished. As such, an early removal of these particles reduces the computation time, as we simply use a distance instead of having to compute the density.

In our selection process, the fluid particles do not move. As such, the contribution of the boundary particles to their density computation is a constant $\rho_{constant\_i}$ that can be computed once and stored. We also have found during our experiments that it is possible to use an empirical rule to quickly eliminate some particles depending on their $\rho_{constant\_i}$. Indeed, during all our simulations we have never seen any particle with a constant density contribution higher than 70% of the rest density ρ_0 of the fluid. We thus added a second early elimination step that removes any particle with a constant density contribution above that threshold.

For the main selection process, we must eliminate particles until the remaining ones properly represent the desired fluid volume and density. As the loaded particles are at rest, any particle with no boundary within its kernel radius will not be eliminated. As such, only a small number of particles is considered in the next steps. In our approach, we use rules to select particles depending on their density as it is fast to calculate and the average is a good estimation to know if we have removed enough particles. As in the SPH approach the particle properties depend on the neighbors, eliminating a single particle has an impact on a large number of particles. We then cannot simply set a desired density threshold and remove every particle above it. Such a process would result in a density lower than the desired one and would create void regions in the fluid. Instead, we use an iterative process that starts with a high density threshold and reduces it while re-computing the particles density at each iteration. We propose the following selection process:

- initialize the selection threshold ρ_{cap_max} with the maximum density observed across all particles
- reduce the threshold by $\Delta_{\rho_{cap\_max}}$
- compute the density of every candidate for elimination
- eliminate any candidate above the threshold ρ_{cap_max}
- repeat the previous 3 steps until the average density of the candidates reaches the target density ρ_{target}

This selection process can be quickly executed as the particles are static so their neighbors do not need to be recomputed. If the implementation is sequential, the density computation needs to be executed only once as each removed particle can subtract its own influence on its neighbors. However, as we use a GPU implementation that needs synchronisation, we did not benefit from this and it may actually lead to higher computation times compared to simply recomputing the density.

3.1.1 Protection rule If the variation between two iterations $\Delta_{\rho_{cap\_max}}$ is too large, large groups of packed particles will be removed in areas where the density is too high. This will create large holes in the fluid. This issue can be solved by using a lower variation $\Delta_{\rho_{cap\_max}}$ as it will only remove the highest particles in the high density areas which will lower the density of all other particles. However, this would greatly increase the computation time. To limit the negative impact of using larger $\Delta_{\rho_{cap\_max}}$, we added a validation step on particles that have been selected as removal candidates for the current step. After having computed the density, any particle with a density above the threshold verifies if it has a neighbor with a higher density than itself. If such neighbor is found, the particle then estimates its density ρ^* without the contribution of this neighbor ρ_{ij}. If ρ^* falls below the current value of the threshold ρ_{cap_max} then the particle is protected from elimination.

3.1.2 Step size regulator Our algorithm relies on the diminution of the target density by a constant step at each iteration. This may lead to a final density significantly lower than the target density if the average density before the final step is just slightly above the target density. The larger the density step $\Delta_{\rho_{cap\_max}}$ is, the more important this problem will be. We can reduce it by reducing the step size the further we progress in the selection process. At each iteration, the error between the target density ρ_{target} and the average density ρ_{avg} should be reduced until it reaches 0 at the last iteration. By linking the step size to this error evaluation, we will maintain a large step size if it does not progress fast enough and can quickly transition to a smaller step size if too many particles are removed at a given iteration. For this, we used a factor of $R_{ssr} = 2$ in all our experiments, i.e. $\Delta_{\rho_{cap\_max}} = (\rho_{target} - \rho_{avg}) \times 2$. This factor results in no loss of quality while improving computation times (see section 5.2). Also, we capped $\Delta_{\rho_{cap\_max}}$ to a minimum value of 0.5% of the fluid rest density to prevent variations

Algorithm 1: Particle selection process

```
Remove particles outside the simulated area and the
  boundary particles;
Load pre-simulated file with boundary, fluid and air
  particles;
Tag loaded particles that have solid or already
  existing fluid neighbors as candidates;
foreach p ∈ candidates ∪ air do
    if solid or existing particle < 0.5 ×
     particleRadius then
        remove particle;
    end
    compute and save ρ_constant_i;
    if ρ_constant_i > 0.7 × ρ_0 then
        remove particle;
    end
end
foreach p ∈ candidates do
    search and store neighbors in fluid;
    compute ρ_air_i;
    compute and save ρ_constant_i += ρ_air_i;
    compute ρ_fluid_i;
    ρ_i = ρ_constant_i + ρ_fluid_i;
end
ρ_avg = Σ ρ_i;
ρ_cap_max = ρ_i_max;
while ρ_avg > ρ_target do
    Δ_ρcap_max =
     max((ρ_target − ρ_avg) × R_ssr, 0.005 × ρ_0);
       // see 3.1.2 ρ_cap_max −= Δ_ρcap_max;
    foreach p ∈ candidates do
        if ρ_i > ρ_cap_max then
            tag as removalCandidate;
        end
    end
    foreach p ∈ removalCandidate do
     // see 3.1.1
        foreach p_j ∈ removalCandidate do
            ρ*_i = ρ_i − ρ_ij;
            if ρ*_i < ρ_cap_max then
                remove removalCandidate tagging;
            end
        end
    end
    foreach p ∈ removalCandidate do
        remove particle;
    end
    foreach p ∈ candidates do
        compute ρ_fluid_i;
        ρ_i = ρ_constant_i + ρ_fluid_i;
    end
    ρ_avg = Σ ρ_i;
end
```

too small to have any significant progress toward the convergence.

3.2 Fluid particle selection at liquid boundary

Our algorithm offers the possibility to load a volume of fluid at rest adjacent to an already existing volume of fluid. The handling of the particle selection process at a fluid-fluid interface is mainly the same as the one at a fluid-boundary interface. As such, the existing fluid can simply be handled as if it was a solid object and its contribution to the density can be computed once and added to $\rho_{constant\_i}$. However, there is a major difference between the existing fluid and any solid object that would be sampled from a mesh. We do not know the geometry of the existing fluid, there is no simple way to remove the loaded particles that are *within* the already existing fluid as it is done for the solid objects. However, our two early selection rules are able to extract all those particles easily. Any particle that is *within* the existing fluid will either be extremely close to the fluid particles or will have a density contribution from the existing fluid close to the rest density as it is a sampling point inside of a fluid at rest. Still, this means that the selection process will have significantly more particles to handle at the early removal step.

3.3 Fluid stabilization

Our selection process gives an average density very close to the fluid rest density. However, starting the simulation from that distribution would lead to movements of the particles as we have variations of densities around that average. To solve that problem we use a localized damped simulation as a stabilization step. Since we know that any new fluid particle that is not in the neighborhood of either the boundaries or the existing fluid that already has a perfect density, we do not need to simulate them and they will be used as fixed particles. We only then have a fine layer of simulated particles that are in contact with a large number of perfectly distributed particles and we have found that such simulation converges extremely quickly. The choice of a damped simulation over a faster algorithm, such as the particle packing algorithm, is motivated by two reasons. The first one is that it does not require any additional implementation as we can simply use the normal simulation loop. The second one is related to the use of the predictive corrective SPH algorithm called DFSPH (Divergence Free SPH). This algorithm can use warm-start values for its predictive corrective scheme to greatly reduce the computation time. As such, the use of a damped simulation allows us to know the values for the warm-start process once the stabilization phase is finished, even if it is only on a limited number of particles. This leads to faster early simulation steps when the actual fluid simulation is executed. Appendix A.1 introduces the DFSPH simulation scheme, but we refer the reader to Bender et al.[22] for its full description.

Similarly to previous works, our damping approach is linearly proportional to each particle velocity, using a coefficient $\alpha \in [0, 1]$. This coefficient is applied to the velocity just before the computation of the new particles positions. As mentioned in[17], the particle distribution reached with damping may not be the same as without. To limit the discontinuity that will exist at the end of the fluid initialization, we reduce the damping coefficient after each simulation step by a multiplicative factor $\Delta\alpha$. The maximum number of stabilization steps is a parameter of our algorithm

and will be denoted as $NbrST\_max$. The effective number of stabilization steps may be lower if we find the fluid to be at rest before reaching the last stabilization step. We will define a fluid at rest if the maximum velocity is below a threshold of vel_{max_target} and the average velocity is below vel_{avg_target}. At the end of the stabilization process all velocities are set to zero to cancel any remaining motion in the fluid.

4 Dynamic window

One interesting aspect of our particle initialization algorithm is that, once the pre-simulated large fluid file is loaded, we can rapidly reinitialize the fluid even if the boundaries or the existing fluid distribution have changed (see section 5.5). This allows us to add fluid in a rest state between two simulation steps while keeping interactive performance. We will apply our system to create a simple dynamic simulation window. The goal of this window is to allow a realistic fluid simulation around a moving object (typically a boat or a character) without having to simulate the whole area where the object can possibly go at some point. Our dynamic window is made of two components. The first one allows the addition of new fluid in the direction of motion and the removal of the fluid in the opposite direction. For this component, we can directly apply our fluid initialization system, as described in section 4.1. The second component is an open boundary that handles the particles transitions through the simulation boundaries. In our case, the open boundary must absorb the perturbation generated inside the simulated area without generating variation of the simulated fluid volume. This component is presented in section 4.2. Those two components are independent from each other and can simply be executed sequentially. Since when we move the simulation window with the first component we have full control of the fluid height and velocity near every boundaries, we do not need to execute the second component for the simulations steps where the simulation window is displaced. In our application scenarios, we considered that the ocean surrounding the simulated areas is always at a given constant height H_{ocean} and at rest. In this work, we aim to propose realistic animations of a fluid around a moving object of interest while keeping interactive computation times. To be able to integrate external perturbations, such as waves coming from outside the simulation window, a more complete open boundary system would be required. Although they offer great animation quality and realism, existing models require extremely specific boundaries models, contrarily to our system.

4.1 Dynamic window displacement

Since the goal of this component is to add new fluid in the direction where the simulation window is moving, it is nearly identical to our initialization process presented in section 3. However, we can adapt the algorithm to reduce its computation time and to increase its stability. Indeed, in this case, we only have to handle a fluid-fluid interface and no fluid-boundary interface.

First, a pre-simulated file describes a fluid at rest for a simulation with the same boundaries as the ones used in the simulation with the dynamic window. We use our initialization algorithm to generate this pre-simulated file from the usual pre-simulated large simulation. This choice is made to reduce the time taken by the adapted version of the initialization algorithm. By using a pre-simulated file that already fits the simulation boundaries, we can greatly lower the number of particles affected by the selection and stabilization processes. Indeed, as the new particles added near the boundaries are already in a rest distribution and fit the boundary, they will not be affected by the two processes. This leaves us with only having to handle the particles that are close to the existing fluid, effectively cutting the number of affected particles in half. Also, as we show in our results (see section 5.3), our algorithm is significantly faster at handling a fluid-fluid interface rather than a fluid-boundary interface, both stability-wise and computation time-wise, making the choice of using a pre-simulated file already fitted to the desired boundaries even more attractive.

Then, part of the existing fluid is removed before applying the selection process. It is necessary to first remove any existing fluid particles that would be outside of the simulated area once the simulation window is displaced. However, it is interesting to remove more particles. As explained in our results (see section 5.3), with our boundary model and simulation method, the fluid particles that are adjacent to the boundary particles are not distributed with a normal rest fluid distribution. Instead, the particles organized themselves on a regular layer around the solid boundaries. This is also most likely the case with any other boundary algorithm as it is likely that the boundary does not represent a normal rest fluid distribution of particles (even for models that do not use particles for their boundaries). As such, it is interesting to extract that regularly distributed particles layer before trying to add any new particles. This will allow us to have a fluid-fluid interface where both sides of the interface will have a normal rest fluid distribution. To remove those particles, we remove any existing fluid particles that are at a distance $D_{minDistToOldBoundary}$ from the simulation boundary. Finally, we also remove any existing fluid particles that are too close to the simulation boundary after its displacement, at a distance $D_{minDistToNewBoundary}$. By doing so, we will not have any existing fluid particle in the neighborhood of a boundary particle after the displacement. This simplifies the selection process as only the new particles can be affected by the selection and stabilization processes. Also, we take maximum advantage of the pre-simulated file that is already fitted to the simulation boundaries. And finally, it prevents the handling of a fluid-boundary interface, which is slower and less stable.

4.2 Fast perturbation absorption

The goal of this component is to absorb the perturbation generated by the object that the simulation window is following. We do not aim to have a fully fledged open boundary component. Our goal is simply to have a system capable of dispersing any wave caused by a moving object reasonably well while staying independent from the boundary and simulation scheme. That way, both our fluid initialization and perturbation absorption will be independent to the used fluid simulation and boundary model. As such, it would be relatively easy to integrate our system to any existing SPH-based approach[23]. Our approach uses two

components to define the inflow and the outflow of the fluid. They can be superimposed on one another to simulate a boundary that is both an inflow and an outflow area.

4.2.1 Inflow To simulate the inflow of the fluid, we use an approach similar to the one proposed by Napoli et al.[24]. Instead of having particles crossing the boundary of the simulated area to be added in the fluid, they create the particles directly in the fluid when they detect that there is enough space near the boundary. To do so, for each particle that moves away from a given distance to the simulation boundary, they look in a cone behind the particle and if this cone is empty they add a new fluid particle. In our approach, we do not need to study the displacement of fluid particles. We sample the area inside the fluid and near the fixed boundary that we know to be stable relative to the boundary and each other. Then, at each simulation step, we check each sampled position for enough space to add a new particle. To decide if there is enough space we simply check if the minimum distance between the sampled point and any fluid particle is below a given distance threshold $D_{inflowThreshold}$ and if the density of the new particle would be below a density threshold $D_{inflowThresholdDensity}$. A benefit of this approach is that all the complexity is in the sampling phase. On top of being able to be pre-computed as long as the boundaries particles do not change, we can generate a sampling distribution where every position is stable even if all positions were to be added at a single simulation step (e.g. if the simulation area is empty). This allows us to handle the addition test of every position in a single parallel step. We already know that the distribution of fluid particles in a rest state has this property and we can use a rest fluid distribution that we have to compute anyway for the displacement of the simulation window.

The last remaining problem is to set the velocity of the new particles. To do so, we use a weighted average of the velocities of the surrounding particles. The weights correspond to the density contribution of the neighboring particles, i.e. $m_j \times W_{ij}$ so the boundary particles are taken into account in this weighted average. The reason is twofold. First, without it, our inflow system will generate a self increasing current when a large enough gap will be detected near the boundary. Second, this technically allows our inflow system to handle inflow currents in our simulation by setting velocities at the boundary particles (even though they are strictly static in the remainder of the simulation). It means that this inflow approach could possibly be used to implement a fully fledged open boundary model.

4.2.2 Outflow Handling a complete outflow system for an open boundary is extremely complex if we do not have any restriction on the boundary model that is used. Usually, the boundary used for a fluid simulation is *solid*, usually due to the use of solid particles or geometries with constant area and mass. The consequence is that the fluid particles are slowed when getting close to the boundary and there is no consistent area near the boundary where we can decide that the particle is now outside of the simulated area. Even if there was such an area, the removal of a fluid particle would create a gap in the fluid. This would require additional systems to keep the fluid continuous, either by modifying the boundary properties dynamically, which is the approach used by Kassiotis et al.[7], or finding a way so that the particles that have been removed still exist in the simulation, which is the approach used in buffer-based open boundaries. For our outflow, we define a system that removes from the simulation the parts of the waves that are above the height H_{ocean} of the area outside the simulation. To do so, we remove the particles that are above this height and at proximity, given a threshold $D_{outflowThreshold}$, of the boundary of the simulation. This system absorbs the upper part of the wave and also reduces the reflection of the part of the wave that is below H_{ocean}, as when colliding with the boundary, the particles of fluid will accumulate and gain a vertical velocity that will move them into the removal area.

5 Results

In our experiments, we used a solely GPU implementation of the DFSPH algorithm based on the implementation found in the SPlisHSPlasH library (version 1.3.1). This implementation uses the single layer rigid-fluid coupling of Akinci et al.[19] (see Appendix A.2) and the variant of Schechter and Bridson[25] for the viscosity. No surface tension algorithm has been used. We increased the minimum number of iterations of the divergence solver from 1 to 3 as it greatly improves the stability of the simulation. The following simulation parameters are kept constant for all experiments: particle radius $0.025m$, time-step $\Delta_t = 3ms$, viscosity 0.02, rest density $\rho_0 = 1000$, density threshold 0.01%, divergence threshold 0.1%. When running the damped simulation for the stabilisation phase the density threshold is relaxed to 0.05% since it is very unlikely that the early steps of the stabilization would be able to reach a high level of accuracy considering the initial distribution of the particles within a reasonable number of iterations. Since we are using predefined distributions of fluid particles, it is possible that the fluid particles randomly fit the boundary or existing fluid particles leading to results that are unrepresentative of a common usage of the algorithm. To alleviate this effect, each experiment is run 51 times and a random offset, within a range of one particle diameter on each dimension, is applied on the precomputed particles set. In our results, we always report the median values. All experiments have been performed on a desktop with an Intel Xeon W-2255 CPU and a Nvidia GTX 2070 SUPER card. An implementation of our algorithm and the code used for our experiments can be found in the following repository: https://gitlab.liris.cnrs.fr/npronost/sph_dynamic_window.

5.1 Algorithm parameters

The parameters of our algorithm must be chosen to reduce the number of iterations for both the particle selection process and the stabilization process. Unfortunately, the optimal parameters for one process may be different for the second process. This could typically be solved by an optimization however for our experiments we simply found values which are robust across a very large range of simulations and giving reasonable computation times. Below are the values used in the following experiments for the fluid initialization.

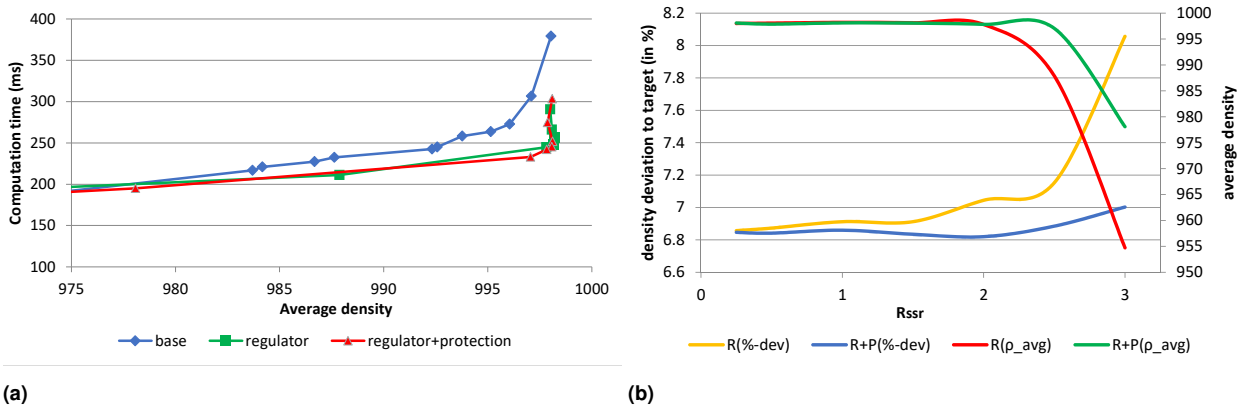

Figure 2. (a) Total computation time (in ms) required depending on the obtained average density (in kg/m^3) for each combination of selection rules. (b) Average density (in kg/m^3) and density distribution relative to target density depending on R_{ssr}.

- $\Delta_{\rho_{cap\_max}}$: controlled by the step size regulator, capped to a minimum value of $\rho_0 \times 0.5\%$ (see section 5.2).
- ρ_{target}: set to 99.9% of the fluid rest density ρ_0. Can be slightly lowered (down to 99% of the rest density) to significantly improve the speed of the stabilization step at the cost of loosing some fluid volume.
- $NbrST\_max$: a value of 10 ensures that the stabilization phase ends with the fluid in a rest state in all our simulations.
- α: is initialized at $\alpha_{ini} = 0.8$.
- $\Delta\alpha$: depends on the initial α value and the number of stabilization steps. It is set so that on the final stabilization step we have $\alpha = 0.1$, resulting in the value $\Delta\alpha = (0.1/\alpha_{ini})^{1/(NbrST\_max-1)}$.
- vel_{max_target}: set to $0.25 \times particleRadius/\Delta_t$.
- vel_{avg_target}: set to $0.05 \times particleRadius/\Delta_t$.

And here are the values that are specific to the dynamic window.

- H_{ocean}: customized for each scenario, usually 1 meter.
- $D_{minDistToOldBoundary}$: is specific to the boundary model, set to $3 \times particleRadius$ in our simulations. The goal is to use the lowest value that produces a continuous one particle layer of fluid around all boundaries.
- $D_{minDistToNewBoundary}$: is specific to the boundary model, set to $1.5 \times kernelRadius$ in our simulations. Usually the kernel radius is fixed to $kernelRadius = 4 \times particleRadius$ in SPH simulations.
- $D_{inflowThresholdDistance}$: set between $0.5 \times ParticleRadius$ and $2 \times particleRadius$. Lower values result in a more consistent inflow of particles at the risk of generating instabilities. Set to $1.2 \times ParticleRadius$ in all our simulations.
- $D_{inflowThresholdDensity}$: set between $0.5 \times \rho_0$ and $1.5 \times \rho_0$. Higher values result in a more consistent inflow of particles at the risk of generating instabilities. Set to $0.85 \times \rho_0$ in all our simulations.
- $D_{outflowThreshold}$: set to $kernelRadius$.

5.2 Particles selection

We propose three systems for the particle selection:

- The elimination rule to eliminate the particles above the density threshold ρ_{cap_max}.
- The protection rule to protect the particles that have neighbors with density above the threshold ρ_{cap_max} and for which an elimination lowers the density below the threshold.
- The step size regulator to adapt $\Delta_{\rho_{cap\_max}}$ depending on the error between the target density and the current average density.

The elimination rule is mandatory for reaching the target rest density while the protection rule and the step size regulator allow larger $\Delta_{\rho_{cap\_max}}$ while preventing the apparition of gaps in the fluid. In this section, we study the impact of the protection rule and the step size regulator.

In these experiments, we use a 3 meter high rectangular column of fluid in a (3.5;5.0;3.5) container with a $\pi/4$ rad rotated $2m$ box floating on top of it (see Figure 3a and Figure 4b). We compare 3 configurations: using only the elimination rule (referred to as *base*), using the elimination rule and the step size regulator (referred to as *regulator*), and using all three rules (referred to as *regulator+protection*). For the *base* configuration, we can obtain various results by changing $\Delta_{\rho_{cap\_max}}$ as there is no step size regulator to control it. For the other configurations, the various results are obtained by changing the value of the ratio R_{ssr} used by the step size regulator. For all configurations, larger values of the aforementioned parameter will result in a larger step size being used which will both lower the quality of the result, mostly visible by a larger error of the average density ρ_{avg} post selection step, and also lower the computation time. The lower computation time is a result of both the lower number of iterations for the selection step and the lower average density, making the stabilization step faster.

To compare the behaviour of each configuration, we study the relation between the obtained average density and the required computation time (see Figure 2a). For the *base* configuration, only a single variation (obtained by using $\Delta_{\rho_{cap\_max}} = 5$) is able to result in the best possible ρ_{avg} with an error of only 0.1% relative to the target density. However, it requires $380ms$ to obtain this level of quality. Both the *regulator* and *regulator+protection* configurations are able to obtain the highest level of quality for much lower computation times, with the best computation times

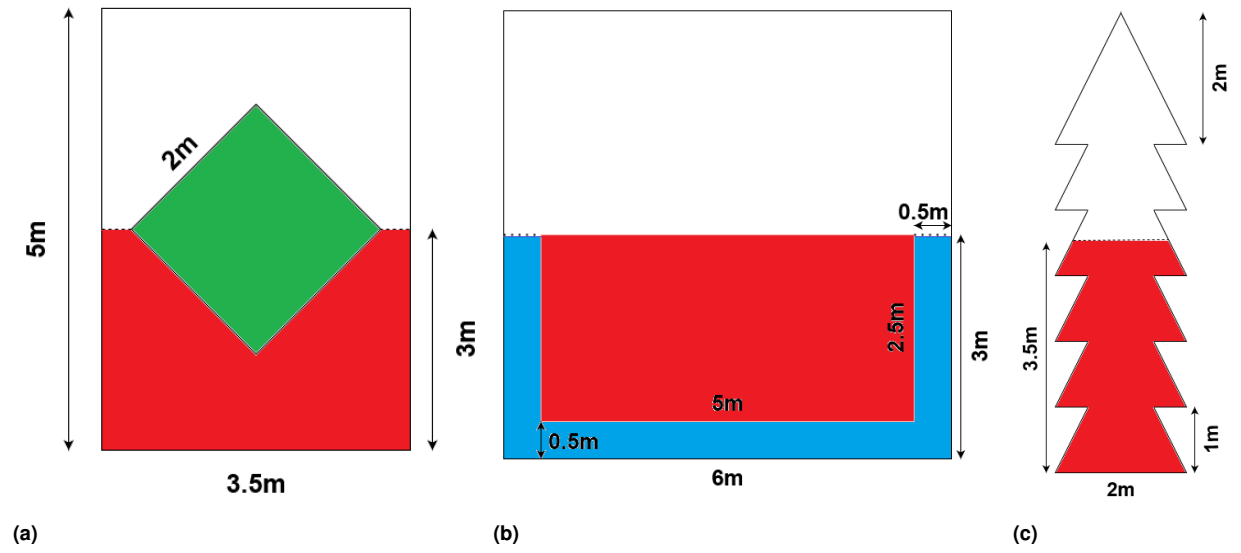

Figure 3. Diagrams of the experiments initialization: (a) floating cube experiment (section 5.2); (b) initialization with pre-existing fluid experiment (section 5.3); (c) complex geometry experiment (section 5.4). The boundaries are in black, the area with a newly initialized fluid is in red, solid objects are in green, and the pre-existing fluid is in blue.

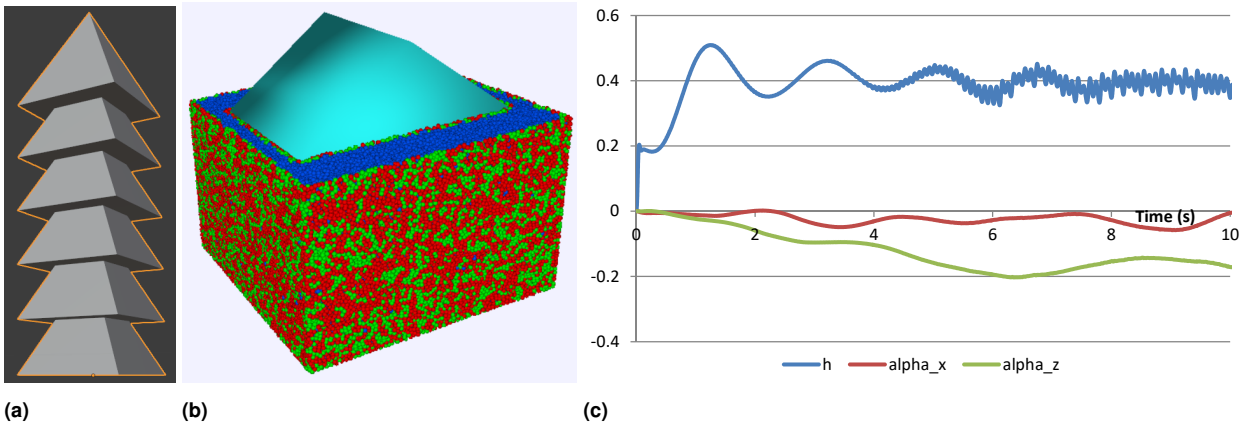

Figure 4. (a) 3D model used as boundaries for the complex geometry experiment. (b) Image of the simulation for the floating cube experiment (the green and red colored particles are active particles that were respectively below and above the density target at the end of the selection step). (c) Evolution of the height (error relative to the expected height, values are relative to the particle radius) and roll angles (in degrees) after the initialization of a box floating above a column of fluid.

being obtained in both cases with $R_{ssr} = 2$ resulting in respectively $244ms$ and $242ms$. These results indicate that the benefit of the protection rule can only be seen if we decide to accept a slightly larger error in the final density. Indeed, the *regulator+protection* configuration is able to produce a result with an error of 0.2% relative to the target density using only $232ms$ with $R_{ssr} = 2.5$. Using the same ratio, the *regulator* configuration fails to result in an acceptable average density and results in an error of 1.2% relative to the target density for a 10% lower computation time ($212ms$). However, when we use a ratio lower than 2.5, the protection rule still brings some benefits. Indeed, using the protection rule results in a better particle distribution at the end of the selection step, that can be quantified by looking at the deviation of the particles densities relative to the target density (see Figure 2b). We observe that having the protection rule on top of the regulator leads to more stable results both in the average density and the density distribution. Indeed, the protection rule is able to maintain a high level of quality for the particles distribution up to $R_{ssr} = 2.5$ compared to using only the step size regulator where the distribution starts to degrade even with $R_{ssr} = 1$. This gain of stability is not visible in the computation times because our stabilization step uses a damped simulation with the DFSPH algorithm not being able to take advantage of better particle distribution. If the stabilization algorithm was changed to a geometric-based one, this particle distribution could also have a positive impact on the computation times of the stabilization step.

Since using the protection rule allows the use of $R_{ssr} = 2$ without getting too close to the range limit and for only a slight increase in computation time, we decided to use this ratio value and the three rules in all our further experiments.

5.3 Particle initialization with existing fluid

To see if there is a significant difference in handling the initialization, we simulated two scenarios : when there are already existing fluid particles in the simulation and when there are none. In the first scenario the affected particles are only in contact with already existing fluid particles, i.e. a

simulation	$N_{particles}$	$N_{NearBoundary}$	$t_{loading}$	$t_{selection}$	$t_{stabilization}$	t_{total}	iter. sel.	iter. stab.
fluid-fluid	1045405	55098(62048)	15.0	109.3	122.1	246.4	6	6
fluid-boundary	605810	49895(62007)	14.1	88.3	124.0	226.4	6	10
pyramid	75142	21222(22963)	15.5	38.0	183.5	237.0	7	10
floating cube	309225	44662(48293)	15.2	74.3	156.8	244.7	7	10

Table 1. Timings (in ms) of the particle selection experiments. $N_{NearBoundary}$ indicates the number of particles near the solid particles after the fluid selection process, in parentheses the number before the application of the main selection loop. The timings shown in this table suppose that the large pre-simulated file was loaded before running the experiments.

fluid-fluid interface, and in the second one they are only in contact with the boundary, i.e. a fluid-boundary interface. For the first one, we pre-simulated a (6.0;3.0;6.0) cube of fluid until it is at rest, then we removed any particle that is within a (5;2.5;5) cube placed at $0.5m$ of each boundary and we used our algorithm to reload that central area while keeping the remaining fluid particles (see Figure 3b). Since we will compare the computation times we need a configuration that has roughly the same number of active particles before the selection process, i.e. affected by the selection and stabilization, in a scenario where we do not keep the existing fluid. This can simply be obtained by having the same initialization surface but in contact with solid particles. In this scenario, we simulate a (5.5;6.0;5.5) cube of fluid at the bottom of a (5.5;2.5;5.5) container. In both scenarios we have around $62k$ active particles before the particle selection iterations.

The results are presented in Table 1. First, for the same number of active particles, the selection process needs to remove much more particles, resulting in around 10% fewer particles after selection in the fluid-boundary scenario (the computation time difference is only due to having $400k$ more particles to load at the end of the selection). We can also observe that the stabilization process can reach the desired velocity targets when dealing with a fluid-boundary interface as it reaches the maximum number of stabilization iterations. In contrast, the experiment with the fluid-fluid interface reaches them after 6 iterations. Although it does not result in better stabilization times, in the fluid-fluid scenario we have 10% more particles and since the particles being stabilized are in the middle of the fluid they have around twice as many neighbors particles resulting in a longer divergence step for the DFSPH algorithm. The faster stabilization is due to a better particle distribution after the selection process. We can evaluate this difference by comparing the standard deviation of the density in both scenario. For the fluid-fluid interface, we have a deviation of 3.4% and for the fluid-boundary interface a deviation of 6.9%. Better results in the fluid-fluid scenario are observed as, when a fluid is at rest, the distribution of the particles near a boundary is significantly different from the one at the center of the fluid. Although this may be unique to Akinci et al.[19] boundary model, it is very likely that this phenomenon exists with most boundary models. The reason is that the boundary particles are distributed on a layer, e.g. for a plane boundary all particles will be contained in that plane. This means that any particle near this boundary will have a large number of neighbors organized on a regular structure most likely causing them to have a relatively regular distribution. In the middle of the fluid there is no particular distribution of the particle and this problem does not exist. The pre-simulated large simulation, that we use to know the positions of the particles we are trying to load, does not have such boundaries. As such, when we load a fluid-fluid interface, we load a center of rest fluid particle distribution and expect the same type of distribution of particles after the stabilization. However, when we load a fluid-boundary interface, we load a center of rest fluid particle distribution and expect a semi-regular distribution after the stabilization which will require more iterations to be achieved.

5.4 Complex geometry test

Following Negi and Ramachandran[18], we used a zig-zag shaped border to test our algorithm over a series of concave and convex geometries. To generate a particular 3D geometry we piled 2 meter squared based pyramids with a height of 2 meters. Each pyramid is placed 1 meter above the previous one (see Figure 3b and Figure 4a). We simulate 3.5 meters of fluid corresponding to 3.5 zig-zag shapes. The timings for this experiment are reported in Table 1. Our algorithm handles this case very well. The main observed limitation with this type of boundary shape is that some particles near the boundary have a noticeable velocity a few steps after the simulation is started, even though they reached a near rest state during the stabilization process. This is most likely due to some fluid particles having a high number of solid neighbors and that we use a relaxed density limit for the DFSPH damped simulation. If the stability becomes the main focus of the simulation, more time can be spent in the stabilization.

5.5 Floating object test

In our last experiment, we replicate the floating object test by Colagrossi et al.[17]. This experiment consists in initializing a fluid with a floating object and observing the stability though the displacement of the center of mass and roll angle of the object. In our experiment, we use a $2m$ box, rotated by $\pi/4$, floating on top of a 3 meter high rectangular column of fluid (see Figure 4b, same setup as the particles selection experiment in section 5.2). The variations of the center of mass, roll angle and pitch angle of the box once the actual fluid simulation is started after the application of our initialization method are reported in Figure 4c. The variation of the height is shown relative to the radius of the fluid particles ($0.025m$ in our experiments) and the angles are given in degrees. We can see that there is no significant variation in the height, which stabilizes around $0.4 \times particleRadius$ away from its initial position, which is within an expected variation due to the start of a physical simulation. The variations of the angles are even less

noticeable as they stay lower than 0.2 degrees. This shows that the slight displacement of the particles we can observe during the first few simulation steps do not significantly perturb the interaction between the fluid and the rigid bodies. Finally, we would like to bring attention to the number of particles and computation time required for this experiment (see Table 1). We can see that this experiment initializes more than $300k$ fluid particles with around $45k$ particles near a boundary in a rest state in around $244ms$, with most of the time taken by the stabilization step ($157ms$). Interestingly, the stabilization step is much larger, nearly 25%, than the time taken by the stabilization of the simple particles cube we used in the previous fluid-boundary experiment even though the floating cube scenario as half the total amount of particles and a slightly higher number of particles affected by the stabilization step. This illustrates a limitation of our damped simulation stabilization step, related to scenarios where the stabilization has to be done in a more constrained space than a simple vertical plane. Finally, we recall that the timings do not include the time taken to read the large pre-simulated particle distribution from the drive since this loading step could be done once at the start of a simulation and reused in different fluid initialization processes.

5.6 Wave absorption

The purpose of the perturbation absorption boundary is to absorb any wave generated inside the simulated area. Basically its effect is to absorb the energy that might be introduced inside the simulated area. To test it, we used a corridor of fluid with a length of $10m$ and a width of $1m$. We initialized the fluid with a depth of $1m$ inside the corridor and then dropped a wedge at the center of the length of the corridor. On one side of the corridor (left side) we use the standard solid closed boundary and on the other side (right side) we use our perturbation absorption boundary with a target fluid depth set to the initial depth of $1m$ (Figure 5a). The goal is to generate the same perturbation on each side of the wedge and then study the difference. The amount of kinetic energy computed as the sum of the squared velocities is reported in Figure 5f. We can see oscillations in the kinetic energy, essentially as we introduce an amount of kinetic energy with the wedge (Figure 5b). Then, every time a wave goes up on either the boundary or the wedge the kinetic energy is converted into gravitational potential energy. And finally the potential energy is converted back into kinetic energy when the wave is reflected by the wedge or the boundary. The first two spikes correspond respectively to the first wave created initially by the wedge (Figure 5b) and to the back-flow which results in the secondary wave due to the gap also created by the wedge (Figure 5c). At around $2.1s$ the kinetic energy is spiking again due to the reflection of the first wave on the boundary. The energy is around 3 times smaller on the side where our perturbation absorption system is applied. It is followed by a small spike caused by the third ripple (Figure 5d). At $4.4s$ we can see the spike caused by the reflection of the secondary wave on the boundary. By that time our perturbation absorption boundary has already removed nearly all the kinetic energy and the bump in kinetic energy is barely visible and after $5.9s$ any motion has nearly stopped (Figure 5e). On the side using the regular boundaries the energy still oscillates after $10s$ and even after $30s$ we observed significant displacements.

At $5.9s$ (Figure 5e), our perturbation absorption boundary has properly reached the initial $1m$ depth of fluid, by going from the initial $46k$ particles (one side), down to $38k$ while absorbing the wave, then back up to $40.2k$ particles thanks to our inflow. It is expected to converge to a lower number of particles as the volume of the wedge has been added. The fluid volume is then not determined by the solid boundary but the target depth, making it robust to very dynamic scenes, as long as the depth does not change drastically. To conclude, we can say that our perturbation absorption boundary worked properly to quickly remove the energy without any loss of fluid even though let us recall that it is not able to absorb a current that would be below the specified fluid height.

5.7 Dynamic simulation window

To demonstrate the capabilities of our dynamic simulation window we chose to simulate a scene with a boat moving at high speed. As this boat has no restriction in its movement direction we chose to use a cylindrical simulation area with a radius of $5m$ and a fluid depth of $1m$ (see Figure 6a). In this example, we constantly apply on the boat a force of $5000N$ balancing its drag force and thus resulting in a constant velocity of $2.9m.s^{-1}$. We obtain a realistic animation of the boat and the fluid where the front of the boat arches upward and the back of the boat dips below the normal fluid surface. We observe the expected wave created by the boat cutting through the fluid and the expected perturbations behind it. We also observe a wave forming in front of the boat due to the relatively flat profile of the boat resulting in many particles being pushed forward instead of to the sides. This simulation includes $750k$ fluid particles and $300k$ boundary particles. Each time-step where the simulation window is not moved takes on average $75ms$, including $3.5ms$ for the perturbation absorption boundary and $71ms$ for the DFSPH simulation. For the time-steps where displacement of the window is required, the window is moved in average by $2.5 \times kernelRadius$, i.e. $0.25m$, and it takes $96ms$, making these time-steps around 2 times longer than without displacement. Also, following a displacement of the window the computation time spikes to around $120ms$ for 5 iterations before going back up to the average step duration. This spike is probably due to both the use of a warm start to initialize the DFSPH iterative processes and because we initialized the new fluid with zero velocities which may conflict with the existing fluid. In all, this simulation runs at an average of $100ms$ with the full speed boat. Even though a short freeze is observed every time the simulation window is moved, it is still acceptable when we consider that the same data structure as our fluid was used instead of an optimized custom data structure for the dynamic window. Also we used our standard parameters for this experiment and we could better fit them to obtain lower computation times.

We used a relatively large area in our test scenario to make it possible to completely preserve the perturbations caused by the boat. It is possible to use our dynamic window for scenarios with a much smaller simulated area while still keeping a visually interesting fluid animation. As an example we can reduce the simulated area to a $2.5m$ radius cylinder (see Figure 6b) to greatly reduce the computation

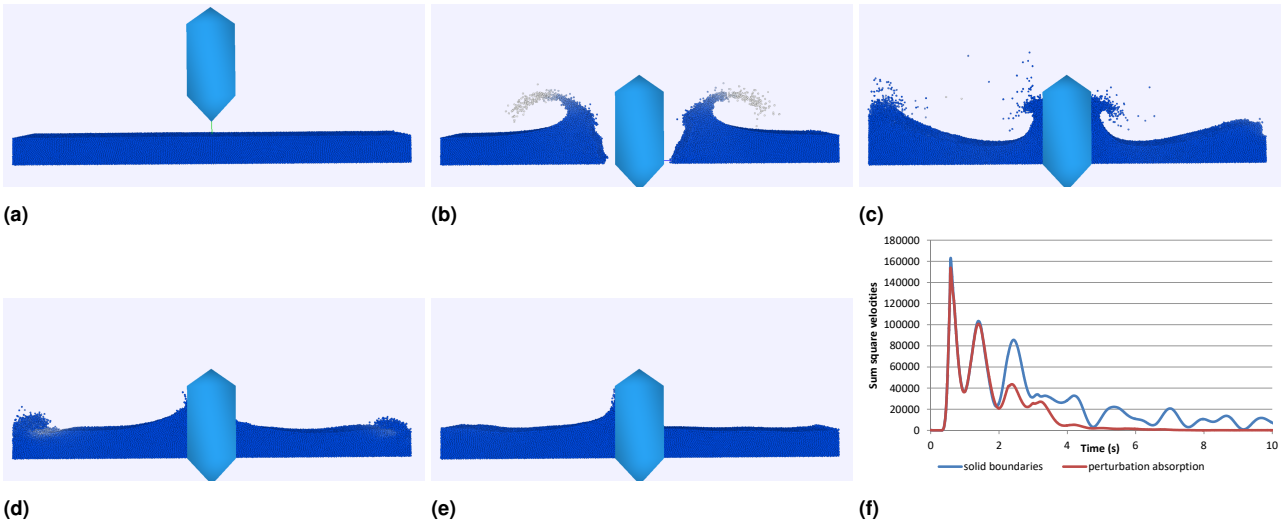

Figure 5. Snapshots of the wedge perturbation absorption test. In order, (a) 0s, (b) 0.9s, (c) 1.7s, (d) 3.6s, (e) 5.9s. (f) Evolution of the amount of kinetic energy, computed as the sum of the squared velocities, on each side of the simulation after the drop of a wedge in the middle of a $10m$ corridor of $1m$ of fluid at rest. In blue, left side of the wedge (normal solid boundary). In red, right side of the wedge (perturbation absorption boundary)

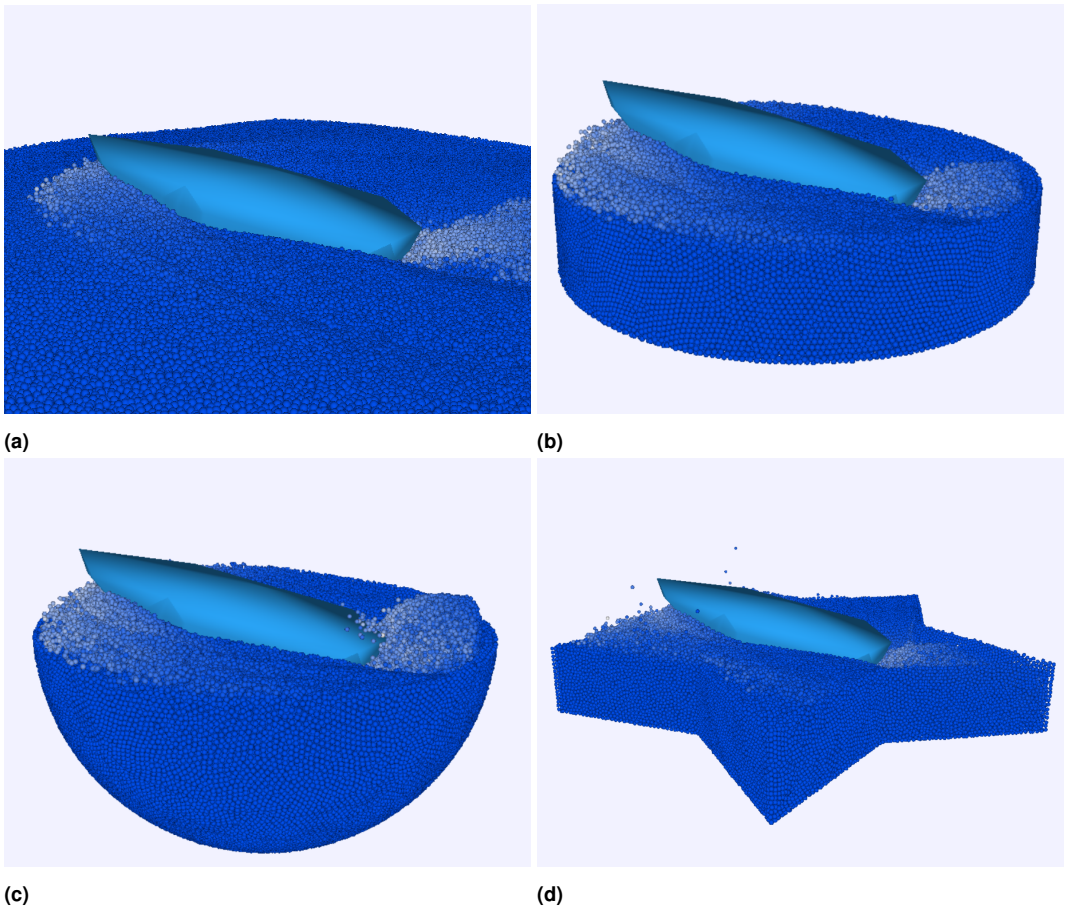

Figure 6. Snapshots of simulations with a high speed boat moving with the use of the dynamic simulation window to keep the fluid simulation around the boat. Our system can handle any boundary shape and here 3 examples are given: 6a cylindrical boundaries (5m radius), 6b cylindrical boundaries (2.5m radius), 6d star-shaped boundaries with vertical walls, 6c spherical boundaries (even the boundary in the air is spherical)

times. With this smaller simulated area (containing $187k$ fluid particles and $114k$ boundary particles), the time steps for which the simulation window does not need to be moved take on average $25ms$ and the displacement of the simulated window takes $40ms$, resulting in an average of $30ms$ with the full speed boat. However, as we can see in the companion video, this smaller simulated area causes a different type of wave to be observed in front of the boat since there is not enough space for the particles pushed to the front by the boat to be accumulated in the same way. Still, the animation is

similar, the main difference being the disappearance of the upfront wave caused by the accumulation of particles due to the boat pushing them forward. Such smaller simulation window could be used to iterate faster on an animation before using a larger window to obtain a higher level of realism. The window size and shape should therefore be decided by the user according to the desired tradeoff between realism and performance. Another way to greatly improve performances as the cost of realism is to aim for a lower ρ_{target} at the end of the particle selection step. As seen in table 1, the majority of the time required to initialize new fluid particles is taken by the stabilization step. By aiming for a density slightly below the rest density of the fluid at the end of the particle selection step, we greatly can lower the complexity of the stabilization step. For example, by lowering the target density to 99% of the rest density we were able to observe a reduction of the number of required stabilization steps from 7 to 3 with the large cylindrical domain. However, using a lower target density means we are relying on our particle inflow system to maintain the number of particles. As we have only used a simple system whose objective is to absorb the perturbation caused by the boat, we noted that using a lower target density resulted in a visible reduction of the fluid level for smaller simulation domains. An intermediate target density could probably be used, but as the exact value would be specific to each simulation domain we have not explored this optimization any further.

In Figure 6 and the companion video, we can notice that when the fluid level is below the ocean level the newly added particles are more clearly visible and could be perceived as a visual artifact. We choose here to visualize how everything happens and to render all particles, but such artifact can be completely removed by rendering only the particles within a distance to the area of interest and/or to the boundaries. In the companion video, we can also perceive the removal and spawning of the particles at the boundaries as a flickering effect but applying any surfacing algorithm to render the liquid will remove this effect, or by simply integrating the window in the larger, for instance procedurally generated, ocean. Another way to mask this flickering effect would be to apply a fading effect to the transparency of the particles. So, instead of appearing and disappearing instantaneously in a single frame, the particles would go through intermediary states where they would be more and more transparent and only visually removed when fully transparent with the opposite being done when particles are added to the simulation.

Finally, our system is not restricted to a simple planar or cylindrical boundary and is able to handle any boundary shape. To illustrate this property, the same scene was simulated with other boundary shapes. First using a sphere (see Figure 6c) and then using a star-shaped prism (see Figure 6d). Although these examples are only used to demonstrate the flexibility of our system, tailoring the boundary shape to a specific simulation can be extremely useful in practice to remove particles that have no interest in the desired animation. For example, we could have used an inverted truncated cone for the boundaries in the boat scene as the particles that are both near the bottom of the fluid and near the borders are not involved in the physics realism of the scene. The simulation using the star boundaries illustrates the problem that can be caused by having the area of interest too close to the boundaries as the fluid animation then differs from the others when the border is too close to the center of the simulated window.

6 Conclusion and future works

In this paper, we presented our work that resulted in two contributions. First, we proposed a novel approach to initialize a distribution of particles corresponding to a fluid at rest and capable of handling complex 3D boundaries. Our method is based on the use of a pre-simulated fluid to initialize the distribution, removing the necessity of modifying the boundaries properties and making it compatible with fluids that would already exist within the simulated area. We used a density-based selection of particles, making our approach theoretically compatible with any boundary model or SPH simulation algorithms. Our approach is fast to compute and scales very well with large simulations.

This paper also presents a novel dynamic simulation window process that allows moving the simulated area to follow a moving object of interest. This process uses a combination of our fluid initialization approach and a perturbation absorption system at the boundaries to prevent the reflection of perturbations generated inside the simulated area. With this dynamic window process, we were able to move a virtual boat around an unrestricted flat ocean while producing realistic looking fluid animations near the boat. Although our method requires a number of parameters and thresholds that is rather high, they are used to fine-tune the simulation to achieve desired results and can be applied to any high-performance interactive applications like video games, but also for prototyping in visual effect software.

From a theoretical standpoint there are two main limitations that can be noted in our work. First, our initialization system uses an average of the density of the particles near the border as an end condition to determine if we have the correct number of particles to fill out the volume near the boundary. However, as the density of the particles depends on their distribution, using the average of the density is a rather lacking, although really fast, way to evaluate the volume. The second one is related to our outflow mechanism. As the removal of the particles is only done above the initially fixed fluid level, it cannot simulate underwater current or integrate a wave generation model to have a more realistic ocean. The use of a complete open boundary, such as Kassiotis et al.[7] model, would greatly open the range of possibilities, although it would most likely significantly reduce the performance. From a performance standpoint, it would be interesting to try to replace the damped simulation by a particle packing algorithm for the stabilisation step as it has been shown to significantly reduce the computation time.

We are currently working on the final surface rendering of both the dynamic window and its interface with the rest of the larger body of water. As our approach does not require any specific fluid and boundary model, we aim to use a fast particles to surface rendering method such as the one proposed by Xiao et al.[26]. In the future, we also hope to improve our dynamic simulation window

and expend its applicability. We would like to expend on our simple perturbation absorption component to make it able to take into account inflow waves at the boundary and handle underwater currents, as possible with some previous works[13], as well as introducing the possibility for a variable ocean target depth. Otherwise this component should be replaced by a proper open boundary model, for example by Kassiotis et al.[7] boundary model. We could also calculate and visualize additional physical fields such as pressure, velocity or density fields. Such display could improve the understanding of the physics involved in the simulations and aid at designing the virtual scenery. It would also be interesting to try to use our dynamic window in a scenario where multiple objects, each having its own simulation window, move in a very large virtual world therefore with multiple dynamic areas of interest. We think our system should be capable to dynamically merge and separate multiple simulation windows by adapting the mass of the solid boundaries when the boundaries of two simulated areas cross each other.

References

1. Stomakhin A and Selle A. Fluxed animated boundary method. *ACM Transactions on Graphics* 2017; 36(4): 68:1–68:8. DOI: 10.1145/3072959.3073597. URL https://doi.org/10.1145/3072959.3073597.
2. Marrone S, Di Mascio A and Le Touzé D. Coupling of Smoothed Particle Hydrodynamics with Finite Volume method for free-surface flows. *Journal of Computational Physics* 2016; 310: 161–180. DOI:10.1016/j.jcp.2015.11.059. URL https://linkinghub.elsevier.com/retrieve/pii/S0021999115008074.
3. Lastiwka M, Basa M and Quinlan NJ. Permeable and non-reflecting boundary conditions in SPH. *International Journal for Numerical Methods in Fluids* 2009; 61(7): 709–724. DOI: 10.1002/fld.1971. URL http://doi.wiley.com/10.1002/fld.1971.
4. Tafuni A, Sahin I, Vacondio R et al. Open boundary conditions for large-scale SPH simulations. In *Proceedings of 8th international SPHERIC workshop*. p. 8.
5. Tafuni A, Domínguez JM, Vacondio R et al. Accurate and efficient SPH open boundary conditions for real 3-D engineering problems. In *12th International SPHERIC workshop*. p. 10.
6. Tafuni A, Domínguez J, Vacondio R et al. A versatile algorithm for the treatment of open boundary conditions in Smoothed particle hydrodynamics GPU models. *Computer Methods in Applied Mechanics and Engineering* 2018; 342: 604–624. DOI:10.1016/j.cma.2018.08.004. URL https://linkinghub.elsevier.com/retrieve/pii/S0045782518303906.
7. Kassiotis C, Violeau D and Ferrand M. Semi-analytical conditions for open boundaries in smoothed particle hydrodynamics. In *Proceedings of 8th international SPHERIC workshop*. pp. 1–4.
8. Joly A, Violeau D, Leroy A et al. Applying Riemann solvers to open boundaries in free surface and confined flows. In *Proceedings of 11th international SPHERIC workshop*. p. 9.
9. Leroy A, Violeau D, Ferrand M et al. A new open boundary formulation for incompressible SPH. *Computers & Mathematics with Applications* 2016; 72(9): 2417–2432. DOI:10.1016/j.camwa.2016.09.008. URL https://linkinghub.elsevier.com/retrieve/pii/S0898122116305107.
10. Verbrugghe T, Dominguez JM, Altomare C et al. APPLICATION OF OPEN BOUNDARIES WITHIN A TWO-WAY COUPLED SPH MODEL TO SIMULATE NON-LINEAR WAVE-STRUCTURE INTERACTIONS. *Coastal Engineering Proceedings* 2018; 1(36): 14. DOI:10.9753/icce.v36.papers.14. URL https://journals.tdl.org/icce/index.php/icce/article/view/8556.
11. Ni X, Feng W, Huang S et al. Hybrid SW-NS SPH models using open boundary conditions for simulation of free-surface flows. *Ocean Engineering* 2020; 196: 106845. DOI:10.1016/j.oceaneng.2019.106845. URL https://linkinghub.elsevier.com/retrieve/pii/S0029801818317633.
12. Chiron L, Marrone S, Di Mascio A et al. Coupled SPH–FV method with net vorticity and mass transfer. *Journal of Computational Physics* 2018; 364: 111–136. DOI:10.1016/j.jcp.2018.02.052. URL https://linkinghub.elsevier.com/retrieve/pii/S0021999118301438.
13. Chentanez N, Müller M and Kim TY. Coupling 3d eulerian, heightfield and particle methods for interactive simulation of large scale liquid phenomena. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '14, Goslar, DEU: Eurographics Association, p. 1–10.
14. Orthmann J and Kolb A. Temporal blending for adaptive sph. *Computer Graphics Forum* 2012; 31(8): 2436–2449. DOI:https://doi.org/10.1111/j.1467-8659.2012.03186.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03186.x. https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2012.03186.x.
15. Solenthaler B and Gross M. Two-scale particle simulation. *ACM Trans Graph* 2011; 30(4). DOI:10.1145/2010324.1964976. URL https://doi.org/10.1145/2010324.1964976.
16. Monaghan JJ. Simulating Free Surface Flows with SPH. *Journal of Computational Physics* 1994; 110(2): 399–406. DOI:10.1006/jcph.1994.1034. URL http://www.sciencedirect.com/science/article/pii/S0021999184710345.
17. Colagrossi A, Bouscasse B, Antuono M et al. Particle packing algorithm for SPH schemes. *Computer Physics Communications* 2012; 183(8): 1641–1653. DOI:10.1016/j.cpc.2012.02.032. URL http://www.sciencedirect.com/science/article/pii/S0010465512001051.
18. Negi P and Ramachandran P. An improved particle packing algorithm for complex geometries in SPH. *arXiv:191007898 [physics]* 2019; URL http://arxiv.org/abs/1910.07898. ArXiv: 1910.07898.
19. Akinci N, Ihmsen M, Akinci G et al. Versatile rigid-fluid coupling for incompressible SPH. *ACM Transactions on Graphics* 2012; 31(4): 62:1–62:8. DOI:10.1145/2185520.2185558. URL https://doi.org/10.1145/2185520.2185558.
20. Bender J, Kugelstadt T, Weiler M et al. Volume maps: An implicit boundary representation for sph. In *Proceedings of ACM SIGGRAPH Conference on Motion, Interaction and*

Games. MIG '19, ACM, pp. 1–10.
21. Hall J, S Rendall TC and Allen CB. Optimisation using smoothed particle hydrodynamics with volume-based geometry control. *Structural and Multidisciplinary Optimization* 2017; 56(6): 1369–1386. DOI:10.1007/s00158-017-1729-x. URL http://link.springer.com/10.1007/s00158-017-1729-x.
22. Bender J and Koschier D. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation - SCA '15*. Los Angeles, California: ACM Press. ISBN 978-1-4503-3496-9, pp. 147–155. DOI:10.1145/2786784.2786796. URL http://dl.acm.org/citation.cfm?doid=2786784.2786796.
23. Shrey S, Kothavale B, Saraf M et al. Smooth particle hydrodynamics: a meshless approach for structural mechanics. *SIMULATION* 2023; 0(0). DOI:10.1177/00375497231180956.
24. Napoli E, De Marchis M, Gianguzzi C et al. A coupled Finite Volume–Smoothed Particle Hydrodynamics method for incompressible flows. *Computer Methods in Applied Mechanics and Engineering* 2016; 310: 674–693. DOI:10.1016/j.cma.2016.07.034. URL https://linkinghub.elsevier.com/retrieve/pii/S0045782516308131.
25. Schechter H and Bridson R. Ghost SPH for animating water. *ACM Transactions on Graphics* 2012; 31(4): 61:1–61:8. DOI:10.1145/2185520.2185557. URL https://doi.org/10.1145/2185520.2185557.
26. Xiao X, Zhang S and Yang X. Fast, high-quality rendering of liquids generated using large-scale sph simulation. *Journal of Computer Graphics Techniques Vol* 2018; 7(1).

A Simulation principles

A.1 Divergence Free Smoothed Particle Hydrodynamics (DFSPH)

The DFSPH simulation scheme[22] is an iterative algorithm that uses iterations inside each simulation step that will iteratively correct the particles so that their physical properties correspond to the desired fluid. In particular, on top of making sure that the density of the fluid stays constant, the DFSPH also ensures that the divergence of the fluid velocity field stays null.

The algorithm 2 presents the organization of a simulation time step of the DFSPH algorithm. The processes to ensure constant density and no divergence are illustrated respectively in algorithms 3 and 4. These correction processes are implemented by corrective predictive loops, which will evaluate the error in the fluid and modify the particle speed to compensate for this error. The error evaluated by these two processes is respectively the error between the average density and the target density and amount of divergence in the fluid.

The velocity corrections for density and for divergence (respectively Δv_i and Δv_i^ν) can thus be expressed as :

$$\Delta v_i = -\frac{1}{\Delta t}\sum_j m_j((\rho_i^* - \rho_0)\alpha_i + (\rho_j^* - \rho_0)\alpha_j)\nabla W_{ij}$$
$$\Delta v_i^\nu = -\sum_j m_j(\frac{D\rho_i}{Dt}\alpha_i + \frac{D\rho_j}{Dt}\alpha_j)\nabla W_{ij} \quad (1)$$

Algorithm 2: DFSPH

foreach *particles i* **do**
  find Neighborhood N_i
end
foreach *particles i* **do**
  compute densities ρ_i compute factors α_i
end
correctDivergenceError(α, v) **foreach** *particles i* **do**
  compute non-pressure forces acceleration acc_i
end
foreach *particles i* **do**
  $v_i \mathrel{+}= \Delta t * acc_i$
end
correctDensityError(α, v) **foreach** *particles i* **do**
  $x_i \mathrel{+}= \Delta t * v_i$
end

Algorithm 3: Constant density solver

Function *correctDensityError(α, v)* **is**
  foreach *particles i* **do**
    predict density ρ_i^*
  end
  while $(\rho_{avg}^* - \rho_0 > \eta) \vee (iter < 2)$ **do**
    foreach *particles i* **do**
      correct velocity $v_i \mathrel{+}= \Delta v_i$
    end
    foreach *particles i* **do**
      predict density ρ_i^*
    end
    compute ρ_{avg}^*
  end
end

Algorithm 4: Divergence-free solver

Function *correctDivergenceError(α, v)* **is**
  foreach *particles i* **do**
    predict density variation rate $\frac{D\rho_i}{Dt}$
  end
  while $((\frac{D\rho}{Dt})_{avg} > \eta^v) \vee (iter < 1)$ **do**
    foreach *particles i* **do**
      correct velocity $v_i \mathrel{+}= \Delta v_i^\nu$
    end
    foreach *particles i* **do**
      predict density variation rate $\frac{D\rho_i}{Dt}$
    end
    compute $(\frac{D\rho}{Dt})_{avg}$
  end
end

The value of the density ρ_i depends only on the position of the particles located in the neighborhood and can therefore be determined at the start of each simulation step. The different variables defined above are calculated according to

the following formulas:

$$\begin{aligned} \rho_i^* &= \rho_i + \frac{D\rho_i}{Dt}\Delta t \\ \frac{D\rho_i}{Dt} &= m_j(v_i - v_j)\nabla W_{ij} \\ \rho_i &= \sum m_j W_{ij} \end{aligned} \tag{2}$$

A.2 Boundary model

In our experiments we used the boundary model presented by Akinci et al.[19]. This model aims to only use a single layer of boundary particles to simulate the presence of multiple layers of boundary. To do so they use a corrected mass value m_{bi} for the boundary particles by computing the volume that each solid particles has to represent :

$$m_{bi} = \rho_0 * V_{bi} = \rho_0 * \frac{1}{\sum W_{ij}} \tag{3}$$

B List of symbols and variables

Symbol	Description	Section
$particleRadius$	Radius of the particles	3.1
ρ_0	Rest density	3.1
$\rho_{constant\_i}$	Contribution of boundary and solid particles to a fluid particle density	3.1
ρ_{air_i}	Contribution of air particles to a fluid particle density	3.1
ρ_{fluid_i}	Contribution of fluid particles to a fluid particle density	3.1
ρ_{avg}	Average density of the particles affected by the selection algorithm	3.1
ρ_{cap_max}	Density threshold for particle removal	3.1
ρ_{i_max}	Maximum density of the fluid particles	3.1
ρ_{target}	Targeted threshold density for ρ_{avg}	3.1
$\Delta_{\rho_{cap\_max}}$	Variation of ρ_{cap_max} between two steps of the selection algorithm	3.1
R_{ssr}	Ratio of the error between ρ_{target} and ρ_{avg} used to compute $\Delta_{\rho_{cap\_max}}$ in the step size regulator	3.1
ρ_i^*	Density of the i^{th} particle without the contribution of a given neighbor particle	3.1
ρ_{ij}	Contribution of the particle j to the density of the particle i	3.1
D	Threshold distance between a particle and the fluid initialization area border for its early removal	3.1
α	Velocity damping coefficient for the damped simulation	3.3
$\Delta\alpha$	Variation of the velocity damping coefficient between two steps of the damped simulation	3.3
$NbrST\_max$	Maximum number of steps allowed for the damped simulation	3.3
vel_{max_target}	Target threshold for the maximum velocity of fluid particles to end the stabilization step	3.3
vel_{avg_target}	Target threshold for the average velocity of fluid particles to end the stabilization step	3.3
H_{ocean}	Fluid height	4
$D_{minDistToOldBoundary}$	Distance to boundaries before displacement of the simulated area	4.1
$D_{minDistToNewBoundary}$	Distance to boundaries after displacement of the simulated area	4.1
$D_{inflowThreshold}$	Threshold distance between existing fluid particles and a particle spawn point to allow a new particle addition	4.2.1
$D_{inflowThresholdDensity}$	Threshold density of a new potential particle to be added	4.2.1
m_j	Effective mass of the particle j	4.2.1
W_{ij}	Kernel value between the particles i and j	4.2.1
$D_{outflowThreshold}$	Threshold distance to boundaries for a particle removal	4.2.2
$kernelRadius$	Radius of the kernel	5.1

Samuel Carensac is currently a research engineer at Ubisoft after being one at the University Claude Bernard Lyon 1, France. He received his Ph.D degree in computer science from the University of Lyon (INSA Lyon), France in 2019. His research interests include fluid simulation, physics-based animation, character animation and AI.

Nicolas Pronost is an assistant professor at the University Claude Bernard Lyon 1, France. He received his Ph.D degree in computer science from the University Rennes 1, France in 2006 after which he worked as a postdoctoral researcher at the Zhejiang University of Hangzhou, China and the EPFL, Switzerland, and then as an assistant professor at the Utrecht University, The Netherlands. His research interests include physics-based animation, soft body deformation, and musculoskeletal simulation.

Saïda Bouakaz received her Ph.D. degree from Joseph Fourier University, Grenoble, France. Currently, she is a Full Professor in the Department of Computer Science, Claude Bernard University Lyon 1, France. Her research interests include computer vision and computer graphics. The current emphasis of her work is the recognition of human motion, gesture recognition and computer animation.