Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2014, Article ID 580154, 12 pages
http://dx.doi.org/10.1155/2014/580154

Research Article

Rigid Body Interaction for Large-Scale Real-Time Water Simulation

Timo Kellomäki

Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland

Correspondence should be addressed to Timo Kellomäki; timo.kellomaki@tut.fi

Received 28 May 2014; Revised 27 September 2014; Accepted 15 November 2014; Published 14 December 2014

Academic Editor: Manuel M. Oliveira

Simulating large amounts of water in real time is often achieved using heightfield methods. Allowing the water to interact with rigid bodies is essential for applications such as games, but traditional heightfield interaction methods concentrate on water-to-body effects by letting water flow through the bodies. We instead take an approach where the bodies block water. Our earlier method is improved in several ways, taking steps toward a single method to create both water-to-body and body-to-water effects. The new method is also visually compared to a traditional method by Thürey et al. A drawback of our method is that it has some grid aliasing artifacts that appear especially when the method is used for floating bodies. However, our method is demonstrated to work together with the Thürey method, which allows us to get the best of both worlds to simulate both floating and blocking bodies in a single scene. The method runs in real time for large areas of water even with a very limited GPU budget.

1. Introduction

Water is a common and important element in nature. It is also frequently encountered in games and other 3D virtual environments. A lot of attention has been devoted to rendering water realistically, but interaction with it is typically limited in games. Bodies of water are often simply modeled as static planes with possibly some procedural waves that have no effect on gameplay and therefore no interactivity. Interaction is sometimes possible with small amounts of particle-based liquids, but they are also more commonly used only for visual effect.

Interactive systems are at the heart of gaming. Off-the-shelf rigid body physics solvers have revolutionized 3D environments in part because they provide so many natural interaction possibilities. Similar interaction with large amounts of water is currently not possible in the real-time 3D realm. A more versatile interaction of rigid body physics and large-scale bodies of water could enrich virtual worlds tremendously and even enable completely new game genres. In the 2D domain, this has already been achieved by games such as Where's My Water [1] and Sprinkle [2] using particles. Some first steps have also been taken in 3D by the pioneering game From Dust [3].

Performance is one of the main reasons why large-scale water areas are still static in most game worlds. Large-scale fully 3D water is still out of reach, but the continuing increases in the available parallel GPU computing power are making some simpler methods fast enough for consideration.

Both Lagrangian (particle-based) fluid solvers, such as smoothed particle hydrodynamics (SPH) [4], and fully 3D Eulerian (grid-based) solvers have been extended to handle interaction with solids in very advanced and realistic ways, with current research focusing more on the pathological cases. In the particle-based fluids, even the real-time results are very impressive for small amounts of fluids [5]. Sadly, neither of these approaches are fast enough for real-time modeling of large bodies of water, such as rivers and lakes [6].

Heighfield-based methods simplify the situation by giving water a single depth value at each 2D coordinate point. This allows them to be much faster than particles or fully 3D methods for large water areas, but a lot of surface detail is lost [7]. From a gaming perspective, an even bigger problem is that interaction with rigid bodies is currently very limited in the heightfield context.

To make interaction possible, water needs to affect the bodies via buoyancy and drag. On the other hand, the bodies also need to affect the water, because they block the flow

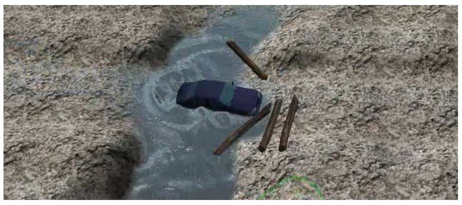

Figure 1: A car crossing a river with floating logs. The car, implemented with our method, blocks water and causes huge waves. The logs are floating bodies implemented with the method of [11] that have a limited effect on the water.

of water, causing waves, and may even make a river take a completely different route.

None of the previous heightfield-based methods solve both interaction directions in a satisfactory way. The traditional solution in real-time methods is to let the water be mostly unaffected by the object, that is, allowing water and bodies to overlap (e.g., [8–14]). In these methods, buoyancy and drag are easy to apply based on the submerged part of the object. Some surface waves may then be generated to create an illusion of the objects affecting the water. However, in these models, water flows through the bodies.

In real world, the objects would naturally push down the water surface. Unfortunately, this breaks the heighfield assumption as soon as water should flow on top of the body. This can be worked around with some limiting assumptions on the blocker shapes or a multilayer model. However, there is hardly any research in this direction, and the few previous methods either have had to let most of the water pass through the bodies [11] or have not been able to include water-to-body effects [15]. The latter method also has problems with rendering.

In this paper, we improve and generalize the method of [15]. This enhances the effect of water on the objects, solves the rendering issues, takes steps toward more general blocker geometry, and replaces some ad hoc solutions by a more physically based approach. The approach and its remaining problems are also analyzed in much more detail than in [15].

Unfortunately, the problem of spatial aliasing from the grid that has plagued the Eulerian 3D fluid-body interaction methods is also present in our method. These problems are mostly negligible for heavy objects but become clearly visible if the method is used to handle floating bodies. As an alternative solution for applications where the aliasing is found to be too disturbing, we propose combining the two interaction approaches. Objects are classified as floaters or blockers. This allows both body-to-water and water-to-body effects in a single scene, as in Figure 1.

All presented methods run faster than real-time on relatively large terrains using our GPU-parallel implementation.

The rest of the paper is structured as follows. Section 2 introduces some related work. Our simulation method, including body-to-water effects, is described in detail in Section 3. Section 4 discusses the problems of adding water-to-body effects in it and suggests a combined method. Section 5 evaluates the results and Section 6 concludes.

2. Related Work

Fluid simulation has a long tradition in engineering. However, most methods from this literature strive for realism, for example, bridge building, and are much too slow for games. In computer graphics, water simulation started to become popular in the 1990s with, for example, the work of Foster and Metaxas [16], but the main application has been in special effects for movies, with processing times for single frames often measured in minutes. We concentrate on methods that are suitable for real-time purposes. For a good introduction on the more realistic approaches, see Bridson's book [7] for the Eulerian approach and the recent review of the SPH literature for the Lagrangian perspective [17].

2.1. Fluid Simulation Methods. Lagrangian (particle-based) methods, such as SPH, are currently popular in the research community [17]. Because the particles are fully 3D, $O(n^3)$ particles are needed to reach a given spatial resolution, which is not yet fast enough for modeling rivers, lakes, and other large-scale features in real time [18]. On the other hand, only the areas containing water need to be simulated, and thus performance is limited by the amount of water, not the total area or volume. Creating a renderable surface also takes a long time compared to heighfield methods [18]. Despite this, particles have been successfully used in 3D game engines for small-scale effects such as leaky pipes [19] and in 2D games even on mobile platforms [1, 2]. Particle-based fluid solvers also have the advantage of having many sophisticated and robust methods for solid-fluid interaction [5, 20–22], though most of them are not designed for real-time applications.

Eulerian methods track the fluid quantities in a fixed grid. While fully 3D methods are mostly too slow for games, there are also faster approaches. Tall cells only model a thin layer near the water surface in 3D, and the underlying noninteresting part is treated as a single, tall cell [23, 24]. This approach yields very high-quality results even in real time but is not yet fast enough for general usage in games, because they have multiple other systems requiring computational resources [6].

The Lattice Boltzmann method has also been used to simulate water [25]. To our knowledge, it is not fast enough for the task of simulating large-scale water in real time due to small time steps. The Lagrangian and Eulerian approaches can also be combined. For example, in movie effect production, FLIP is currently a very popular method, which combines aspects of both methods [26]. As a fully 3D method, it is also too slow for most real-time games.

In open seas without a need for dynamic flow over rough and partly dry terrain, wave particles [12] and FFT-based methods [27] can be used [18]. They can provide relatively realistic-looking results and interaction with rigid bodies, for example, ship wakes [10].

The most relevant methods for our discussion are based on heighfields. Only allowing a single height value for the water surface removes many interesting surface effects, such as breaking waves and splashes, but has the great advantage of making the simulation $O(n^2)$ for a given spatial resolution.

Furthermore, these methods have no need for an expensive linear system solver unlike the Eulerian simulations. Thus heighfield methods allow for the simulation to cover vast areas in real time. Many of the missing surface phenomena can be created using fast procedural methods and only near the viewer, because they are mostly aesthetical and not essential for gameplay.

The pipe method is a very simple and fast water simulation method [8, 28]. It has been extended for multilayer situations [29] and used for purposes such as erosion simulation [30, 31]. Many GPU implementations have also been presented, demonstrating its speed on modern hardware [6, 30, 32].

Shallow water equations provide a more physically based approach to heightfield water simulation [7]. They are slightly more complicated than the pipe method but still fast enough for games. They have also been extended in various ways and implemented on the GPU [11, 13, 33]. However, there is some evidence that the visual difference to the pipe method is quite small when modeling large-scale water bodies in a gaming context [34].

2.2. Interaction with Solids. Simulating the interaction of fluids and solids has been the focus of much research in the fully 3D fluid simulation literature. Lagrangian methods typically model the solids also as particles and are able to produce impressive results even in real time [5]. However, our Eulerian method does not have much in common with these methods.

In the Eulerian context, Foster and Metaxas voxelized the boundary conditions to the simulation grid [16]. Rigid bodies were later handled by Takahashi et al. [35] by rasterizing the rigid body velocities to the grid and setting these as velocity border conditions to the fluid. Many methods run the fluid and rigid body solvers alternatingly, but the quality can be improved by solving both problems simultaneously, as in [36].

The approach of rasterizing border conditions to the grid has problems with boundaries that are not aligned to the grid [37]. A large body of research has addressed these problems since then. A common approach based on the immersed boundary method [38] is to average solid properties in each cell (or dual cell) of the grid and use these as weights in the solver [37, 39]. Some authors also started to model the solids directly in a Lagrangian framework instead of rasterizing them to the grid [36]. A different approach, mostly unsuited for real-time applications, is to give up the grid regularity either completely or at least near the solids [23, 40].

The fully 3D methods have lately been applied to real-time situations, but only on rather small grids. Many of the methods can just as well be applied on the 2D grids, which would be large enough for many games, but this requires the solids to also reside in a 2D domain. In a heightfield situation, the water and the rigid bodies still reside in a three-dimensional domain, but the fluid simulation grid is two-dimensional. Most methods, such as that of Batty et al. [37], are based on modifying the pressure projection step, which does not even exist in heightfield simulations. Because of these reasons, the sophisticated 3D fluid-body interaction methods cannot be directly applied when the underlying simulation is based on heightfields. Thus the heightfield methods have traditionally taken a completely different, and much simpler, approach to the problem.

The heightfield water-body interaction research has mostly concentrated on handling floating objects [8–10, 14]. In these methods, the bodies are superimposed on the water, which makes buoyancy and drag almost trivial to calculate. The effect of the bodies on the water is handled by, for example, slightly pushing down the surface at the bodies, which creates believable waves and ripples for floating objects. The problem of cell aliasing is also present in these methods, and the iWave method solves it similarly to, for example, [37] by estimating an opacity value for the blocker coverage in each cell [10]. Some researchers even calculate interaction per mesh triangle, subdividing the triangles if necessary [12, 13].

However, the previously mentioned heightfield methods are unvariably only interested in the interaction of the solids with the water surface, completely ignoring the body of the water. Approaching water waves are passed through the bodies almost unaffected, which makes sense for small and floating objects that only move up and down with the waves. On the other hand, large or heavy bodies are not handled in a sensible way. This problem is most evident if a body is lying on dry ground and a wave meets it (see Figure 8(a), or the video in the Supplementary Material available online at http://dx.doi.org/10.1155/2014/580154), or when one tries to build a dam out of some heavy bodies. A method that is also able to handle submerged objects was introduced in [11]. It is still mostly interested in surface effects and does not block flow. A method that allows large bodies to block flow by forcing water to flow above, below, or around the body was introduced in [15] but had several shortcomings that we try to address here.

3. Simulation Method

This section describes our simulation method, which is a variant of the classic pipe method of [8] with a layer structure similar to that introduced in [29]. This work is a continuation to [15].

We use a top-down approach, first introducing the general structure of our model in Sections 3.1 and 3.2. A detailed description of each phase follows in Sections 3.3–3.6. Finally, we discuss the parameters and stability in Section 3.7.

3.1. Model Structure. Water behavior is simulated on a uniform 2D grid on the xy-plane. The distance between neighboring cell centers in both directions is ℓ. The model consists of cell-sized columns of terrain, water, blocker, and air stacked on top of each other. A column i occupies the vertical interval (z_-^i, z_+^i). The *depth* d_i of a column is the vertical length of the interval it occupies. A column with zero depth is considered not to exist. Blocker columns are nonpenetrable to water and are used to represent the moving rigid bodies, which are rasterized to the grid resolution.

The model is generalizable to allow any number of water and blocker columns stacked on top of each other, but in

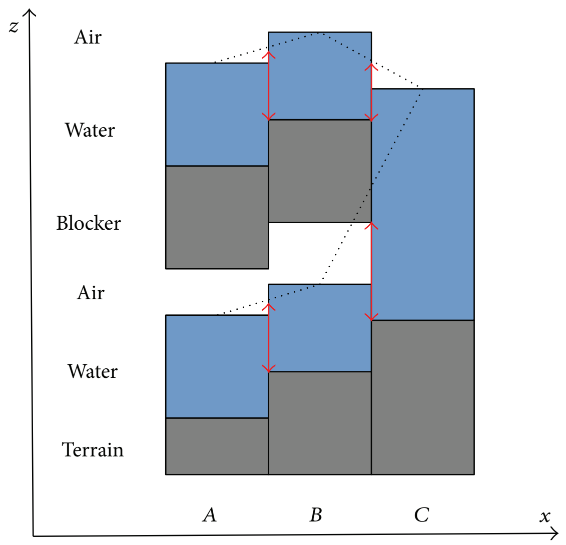

FIGURE 2: A side view of the model structure. Cells A and B have a blocker column with water and air also below the blocker. Cell C is free and thus has only a single water column. Intervals that are used for flow calculation are marked with red arrows. The top of the interval is the average of the two water surfaces. The blockers are taken into account; for example, the bottom flow between B and C is partly blocked.

practice we have found it enough to have a single terrain and a single blocker column plus up to two water columns (one above the blocker, one below), with an implicit air column above each water column. A more general structure would greatly increase the memory bandwidth requirements. The bottom column is always a heightfield terrain that occupies the interval $(-\infty, b(x,y))$. Instead of storing the top and bottom coordinates of water columns, a single height (in meters) is stored for each. When needed, the coordinates can be calculated based on the terrain and body locations. The water *surface* $h(x,y)$ is a heightfield that is constructed by finding the top of the topmost nonempty water column at each cell.

The blocker columns represent the rigid bodies and are found using the rasterization hardware as will be described in Section 3.3. A cell is free, if there is no blocker column. Free cells only have a single water column and a single air column. The water columns are combined by removing the air column between them. A water column is free if there is no body above it. Figure 2 illustrates the structure of our model.

Water is allowed to flow between neighboring cells in the grid. The four von Neumann neighbors are used, but speed could be traded for quality by using the Moore neighborhood. If both cells have a blocker, we assume the blockers come from the same body. This is reasonable, because it is extremely rare that different bodies are located in neighboring cells but are not overlapping. Overlapping bodies would be treated as one anyway, because we only have a single blocker column.

The previous assumption allows us to only store two flows for each pair of neighboring cells: "upper" and "lower." If both cells are free, upper contains the flow, while lower is 0. If exactly one of the cells is free, the two stored flows are

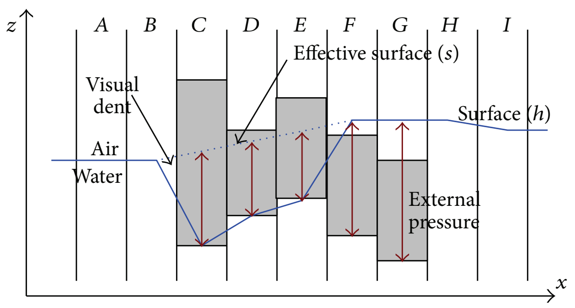

FIGURE 3: A side view explaining surface, effective surface, and pressure. Surface, the top of the topmost water column, is pushed down by the body. Effective surface is an estimate of where the surface would be without the body and is calculated by adding the external pressure to the surface (see Section 3.6). These two surfaces only differ in cells where the surface is pushed down by a body.

free ↔ *top* and *free* ↔ *bottom*. If neither of the cells is free, upper contains the flow between the top layers and lower between the bottom layers.

We define $N(i)$ as the set of neighbors of water column i. The flow between columns i and j is $f_{ij} = -f_{ji}$ (naturally only one value is stored). If $f_{ij} \geq 0$, it is called *outflow* from cell i and *inflow* otherwise.

The external pressure p^i_{ext} at the top of each water column i is also included in the model. External pressure is the result of a rigid body over the column and is zero for free water columns. Air pressure is ignored as negligible. The pressure $p(z)$ inside a water column is assumed to behave according to the basic hydrostatic equation, $p(z) = p(z_+) + \rho g(z_+ - z)$, where $p(z_+)$ is the pressure at the top of the column, ρ is water density, and g is acceleration by gravity. We always give pressure as $p/\rho g$, that is, depth (in meters) of a water column that would cause the pressure. The pressure caused by a rigid body above a water column is modeled as explained in Section 3.6.

Since in this method water pushes the surface down, buoyancy and other forces affecting the bodies cannot be calculated directly in the usual way. Due to the finite grid size, rendering the surface directly would also cause disturbing dents near the objects as illustrated in Figure 3. To solve these problems, an effective surface $s(x,y)$ is constructed to approximate where the surface would be if there were no blockers. The effective surface is another heightfield with value equal to the actual surface plus the external pressure at the surface (in meters). Figure 3 illustrates the concept.

3.2. Algorithm Phases. The following phases are executed to advance the simulation for a single time step of Δt:

(1) rigid body update (by an external physics engine),

(2) blocker update (apply effects of the moved bodies to the water, Section 3.3),

(3) flow update (Section 3.4),

(4) depth update (Section 3.5),

(5) pressure estimation (Section 3.6).

Each step is described in more detail below, but let us first give an overview of the steps. Our method is based on alternating rigid body and water update steps. The rigid body update is a standard simulation step of an off-the-shelf physics engine, Bullet [41] with buoyancy, and other fluid forces applied to the bodies. After that, the water overlapping with the moved bodies is removed, as described in Section 3.3.

Somewhat analogously to the simplifying assumption of hydrostatic pressure always used in the pipe method, we also assume that the rigid bodies are static when solving for water flow and depth. The solid interaction in flow and depth update steps (Sections 3.4 and 3.5) can thus be seen as generalizations of how terrain is typically handled in the corresponding steps of the pipe method, only with several layers and thus the possibility of water flowing both above and under the blocking body. This departs from the more physical approach of the 3D methods, where solid velocities are taken into account, but we have found the results to be quite acceptable.

The final step, pressure estimation (Section 3.6), is needed for visualization and estimating buoyancy and drag.

The method is designed to be completely parallel on the GPU with no random write access. Each phase uses double buffering, only reading the previous state. This ensures that the fast texture memory can be used.

For clarity, no time indices are used for the variables. All formulas are given in an assignment form instead.

3.3. Blocker Update. The blockers are rigid bodies that have possibly moved in phase 1 of the algorithm, which will affect the water.

For each cell, the rasterization finds the bottom and top height coordinate of any rigid bodies overlapping it at the center. This is implemented by rasterizing the triangle meshes using an orthogonal view from above and below. The depth buffer can then be employed to find the extremes, which makes the process very fast even for complicated bodies. See Figure 4 for a visualization of the concept.

In the rasterization step, multiple bodies located on top of each other are combined to a single blocker column. Bodies completely above water are ignored. Otherwise, water between an object even high above the water and a submerged object would be removed.

The next step is to ensure that the bodies and water do not overlap. The blocker columns before and after the time step are handled as being completely unrelated. Note that the vertical extent in a cell blocked by the same body can change radically even during a single time step as the body moves and rotates.

In each cell, the procedure is implemented as follows. The possible blocker column that existed before the time step is first removed. Water is added so that the surface is moved to where the effective surface was. Then the possible new blocker column is added, removing any overlapping water. The addition divides the water column to (possibly empty) parts below and above the blocker. The external pressure in the column below the body also needs to be updated

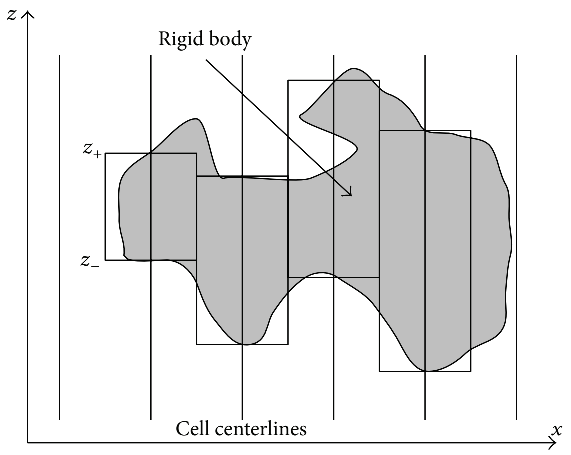

FIGURE 4: A side view of rasterizing the rigid bodies to the simulation grid. In each cell, the vertical overlap (z_-, z_+) of the body and the cell centerline is found. This particular body overlaps 4 cell centers to varying vertical extents.

so that the effective surface is not changed (see pressure approximation in Section 3.6).

During the whole process, the depth of artificially added and removed water is kept track of. Their difference could be either positive (water was added) or negative (water was removed). To conserve volume, the same amount needs to be removed from or added to the vicinity. This is achieved by averaging the amount of missing or extra water with the four neighbors on each time step. Whenever a free cell has a value indicating missing or extra water, a portion $0 < \alpha \leq 1$ of the difference is added or removed. This is what mostly causes waves around a body dropped into water. Our approach is very similar to that used in [11].

An example of this phase can be found in Figure 5. For more details, see [15], which has a similar implementation.

3.4. Flow Update. In addition to removing water from inside the bodies, it is also essential to prevent any flow from entering the bodies. To achieve this, we let the cross section of the pipes connecting columns to vary, similarly to [29], but taking blocker geometry into account.

The flow between two water columns i and j that are neighbors goes through an interface that is represented by the red arrows in Figure 2. The bottom of the interface $z_-^{ij} = \max(z_-^i, z_-^j)$. For the top, we first interpolate the surface height at the interface by taking the average of the tops of the water columns (dashed lines in the figure). We also find the bottom z_-^b of the lowest blocker above the interacting water columns, which sometimes limits the interface (e.g., lower flow between cells B and C in Figure 2). Putting these together, the top of the interface is $z_+^{ij} = \min((z_+^i + z_+^j)/2, z_-^b)$. The cross section of the interface is $c_{ij} = \ell \cdot (z_+^{ij} - z_-^{ij})$.

Flow is caused by a pressure gradient. In column i, the pressure relative to the bottom of the interface z_-^{ij} is $p_i = z_+^i - z_-^{ij} + p_{\text{ext}}^i$, with p_j calculated similarly. The pressure difference

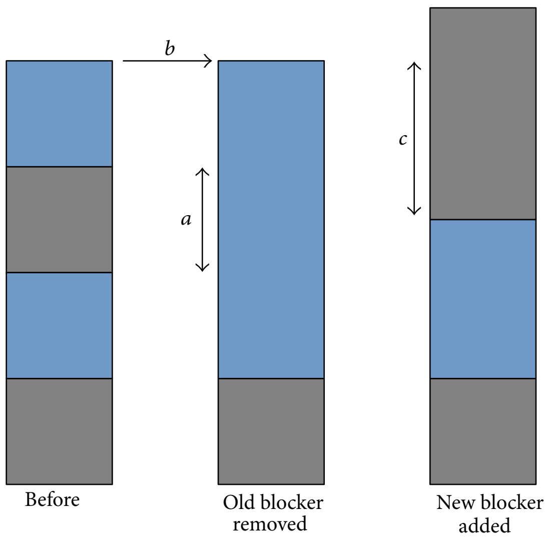

FIGURE 5: An example of the body update in a cell. During a time step, a submerged body rises above the surface. This is handled by first removing the old blocker. In this case, water needs to be added (a) to keep the surface unchanged (b). The new blocker is then added, removing any overlapping water (c). In this case, the upper water column becomes empty.

is $\Delta p_{ij} = p_i - p_j = z_+^i + p_{\text{ext}}^i - (z_+^j + p_{\text{ext}}^j)$, analogously to the original pipe method of [8].

The flow can now be updated using an explicit Euler step as in the standard pipe method:

$$f_{ij} := \mu f_{ij} + \Delta t c_{ij} \frac{g \Delta p_{ij}}{\ell}, \tag{1}$$

where $\mu \leq 1$ is a friction coefficient.

Finally, the flow from each column is restricted to prevent the depth from becoming negative. This is done by calculating a scaling factor $K_i = \min(1, d_i \ell^2 / (\Delta t \sum f_i))$, where $\sum f_i = \sum_{f \in N(i)} f_{ij}$ is the net outflow. Each outflow from i is then multiplied by K_i.

3.5. Depth Update. The depth of each column can now be updated based on the flows. The flow through each interface in a single time step is limited to the depth of the interval, d_{ij}. This step is not physically based but takes care that intervals that have become almost or completely blocked do not let water through due to previously accumulated flow. It is therefore a key element of making the bodies block flow. For example, water cannot flow on top of a body before the surface is high enough. This is roughly analogous to how terrain is handled in the related methods (e.g., the reflective boundary conditions caused by high terrain in [13]). After this limiting, the update assignment is

$$d_i := d_i - \Delta t \frac{\sum f_i}{\ell^2}. \tag{2}$$

Finally, some water columns are under a blocker. If this update would cause a water column to overfill so that it would overlap the blocker above, only water equal to the free portion is allowed to flow. Experimentally, we found out that stability is greatly increased by also decreasing the incoming velocities to such a cell, which can be thought of as a crude model of a no-slip border condition with the blocker. If $d_{\text{overlap}} \geq 0$ is the depth of the overlap and d_{total} is the total incoming depth (not net incoming depth) during a time step, we multiply all inflows to the cell by $(d_{\text{total}} - d_{\text{overlap}})/d_{\text{total}}$. This approach also removes the artifacts caused by the need to slow down underflow in [15].

3.6. Approximating Pressure. The effective surface $s_i = h_i + p_{\text{ext}}^i$ is needed for buoyancy and rendering but is not directly available in places where a body pushes the surface down. Figure 3 is referred to as an example throughout this section to help understand the situation.

It would be possible to track the pressure under a body using physical simulation and calculate the effective surface that way. Instead, we use a simpler approach, and interpolate the effective surface from around the bodies. The external pressure is zero in the free columns, which allows the surface to be used directly as the effective surface (cells A, B, and F through I in the example). These are the border conditions of the interpolation.

In practice, the interpolation is achieved by applying a blur filter on the effective surface of the previous time step and storing the external pressure calculated from that. The calculated values are only used in the nonfree columns. In Figure 3, the results are applied to cells C–E. As a few time steps go by, the effective surface in these cells gradually rises from the surface, approaching the dashed line in the figure.

However, this process is complicated by the bottom water columns (in F and G). Their external pressure could be calculated directly as the difference to the surface. Now imagine the water level at F changing slightly so that the top water column in it dries up. This would suddenly push the effective surface below the body, which would result in visually disturbing flickering and unstable buoyancy. We work around this by keeping track of the pressure also in the bottom water column of such cells and letting the external pressure be the maximum of the directly calculated value and the iteratively solved value.

Formally, let i be a nonfree column and A the cell it is located in. The current external pressure p_{ext}^i, stored in each such column i, is updated during a time step:

$$p_{\text{ext}}^i := \max\left(\left(\kappa s(A) + \frac{1-\kappa}{4} \sum_{B \in N(A)} s(B)\right) - z_+^i, h(A)\right), \tag{3}$$

where $N(A)$ is the set of neighboring cells of A and κ controls the spreading speed of pressure.

The maximum in (3) affects cells such as, for example, F, where $h(F)$ is the top of the top water column. While the body in F is underwater, the external pressure keeps the effective surface at $h(F)$, but now, after F dries up, the external pressure stays. As a result, the effective surface changes very smoothly, which is essential to keep the visual appearance continuous in time.

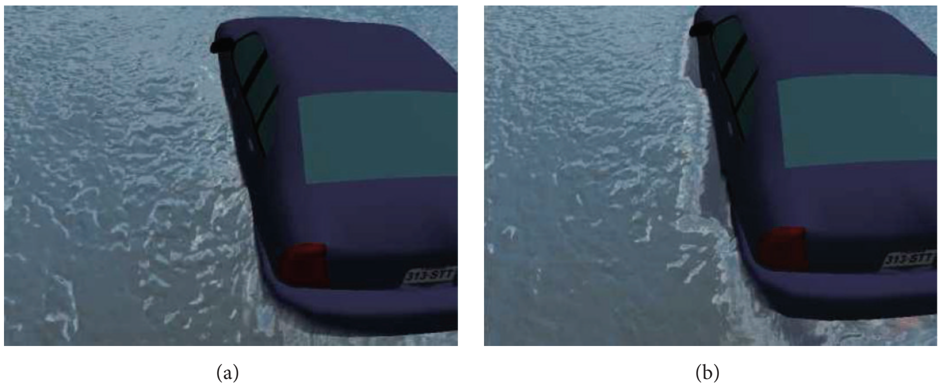

Figure 6: (a) Using the effective surface results in a flawless water-object border. (b) Dents and grid artifacts are visible when the surface pushed down by the car is used.

For a visual comparison of the surface and the effective surface, see Figure 6 and the accompanying video.

3.7. Parameters and Stability. As a variant of the pipe method, our solver also uses explicit time integration. This sets a limit to the time steps that can be used before disturbing instabilities are experienced.

The proposed method has a few parameters that affect the stability and other properties of the simulation: μ, Δt, ℓ, α, and κ. The stability properties are hard to analyze exactly due to the complexity of the blocking, but the idea of the CFL condition $\Delta t \leq \ell / v$ [7] also carries over to our method: to make the method stable, a short time step is needed to compensate for large wave speeds (where wave speed is measured in simulation cells per second).

A longer time step makes the simulation use less computational resources. While the time step might affect the simulation in some subtle ways, a previous user study did not find any noticeable effect when varying the time step inside a stable range in a similar simulation without the rigid body interaction [34]. Therefore, after fixing the values of the other parameters, the time step should be chosen as large as possible without making the simulation unstable.

The friction parameter, μ, describes how much of the old flow is conserved between time steps and should therefore be a function of the time step length. A smaller value makes the water appear more viscous but also limits the velocities accumulated during the simulation, therefore allowing longer time steps. The simulation scale is controlled by ℓ. A smaller ℓ (a finer grid resolution) needs to be compensated with a smaller time step or smaller velocities.

A larger α causes water displaced by the moving bodies to be added back faster, causing bigger waves to emanate from a blocking body in the water. This has the side effect of causing spikes and large flow velocities if the value is too large. Finally, a larger κ makes the gaps near objects disappear faster but also makes the effective surface look more twitchy, since it reacts faster to blocker geometry changes.

Typical values for the parameters in our simulations are $\mu = .999$, $\Delta t = 20$ ms, $\ell = 1$ m, $.01 \leq \alpha \leq .1$, and $\kappa = .5$. With these values, the stability problems have been minimal.

4. Water-to-Body Effects

The method described thus far does not implement any effects caused by the water to the bodies. These effects can be divided into two classes: drag and buoyancy. By drag we mean that water and object velocity tend to equalize. This is especially important in the case of floating bodies, such as logs, that need to follow the flow. Both of these effects are easy to achieve if the water surface is allowed to go through the floating object. In our model, the actual surface is pushed down by the object, but the effective surface can be used in its stead. Our approach here is fairly standard but is shortly described for the sake of completeness.

For the drag calculation, we additionally need a velocity v_A in each cell A. Exactly as in [30], the x component can be estimated by calculating the average flow in the x direction divided by depth of the column in question and y correspondingly. Only the top column is used since drag is mostly relevant for floating bodies.

Drag is based on iterating through all cells overlapped by the body. For each cell A, we first calculate u_A, which is the the vertical extent of the body below the effective surface. We also calculate the local velocity v_A^b of the rigid body in the center of the cell at the middle of the underwater part. The drag force $F_A^{\text{drag}} = (1/2) C \rho u_A \ell (v_A - v_A^b)^2$, where C is a drag coefficient that can be set empirically according to the needs of the application. In our scenes, $1 \leq C \leq 15$. The force is applied at the middle of the underwater part during each time step.

Buoyancy is equal to the weight of the water displaced by the body, again applied in each cell overlapped by the body. We reuse u_A from above and for each cell apply $F_A^{\text{buoyancy}} = -\rho g \ell^2 u_A$ at the center of the cell.

One problem in this approach is that the drag and buoyancy forces are calculated in only the grid locations, using relatively long time steps, which often causes excess rotation. This can be fixed by simply damping both the linear and rotational velocities of the bodies using an empirically determined coefficient based on the proportion of the body that is underwater. This works relatively well in practice but is not physically based. However, our implementation of

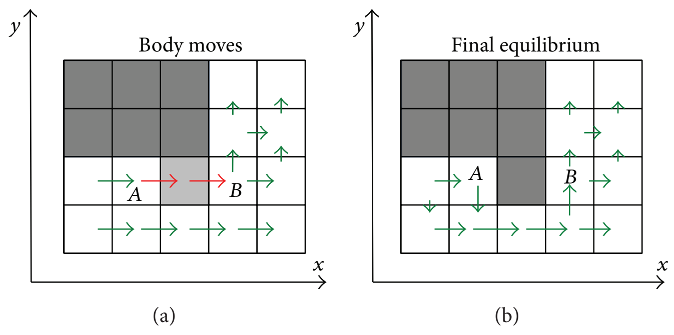

Figure 7: Top view of an aliasing problem. A body (dark gray) is staying in a stable current. Arrows show the flow between cells. The blocker slightly moves to occupy a new cell (light gray), causing the red flows to instantly become blocked. Cell A now has much more inflow than outflow, which causes a sharp peak in the surface before the flows settle down to a new equilibrium (on (b)). Similarly, the surface in B will suddenly drop. Both of these cause large waves to emanate from the area.

the method presented in [11] also needed to resort to similar tricks.

4.1. Aliasing Problems. The suggested blocking method (and that of [15]) is binary: each cell is either blocked or non-blocked. As explained in the related work section, the problem is a common theme in the Eulerian fluid simulation literature. In our case, where the quality goal is much less ambitious, this problem is usually negligible. However, a very small change in the body position can cause relatively large changes in the rasterized version, when a body suddenly overlaps a cell center. This spatial aliasing can cause overly large effects to the water that become noticeable if the body is only moving slowly.

The blocking method is thus best suited for situations where the bodies mainly move independently of the water. If a body stops completely, the physics updates can be disabled and the aliasing problem disappears. On the other hand, applying even an otherwise insignificant buoyancy force to a heavy object keeps the body alive, which causes small vibrations in most rigid body solvers and makes the problem visible. How disturbing one finds this phenomenon depends on the scene and parameters.

Unfortunately, one of the worst cases is an object floating along a strong current. When a cell is suddenly blocked, the stable flow to the cell is denied, causing the source cell to be quickly overfilled. This small movement causes a comparatively large wave to emanate from that cell to the surroundings. This looks unnatural, since there is almost no relative movement between the body and the water, meaning no waves should be created. The problem is illustrated in Figure 7.

Several directions for a solution were examined in our research process. Simply limiting the maximum velocity or making the grid denser makes the effect smaller but does not solve the problem without making the method too slow. The best simple workaround we found is limiting the amount the surface in any cell can change in a single time step to some constant, but it only makes the problem somewhat more apparent.

A better solution could be based on the fractional cell method, such as in [37] or [10]. However, they are not trivially applicable, as trying to take the fully 3D nature of the problem into account is problematic with our underlying 2D fluid solver and trying to address the full complexity of the 3D situation would quickly become too slow. A strictly 2D solution such as in [10] also does not work as such in our context, because our method completely relies on the vertical intervals to block the flow. In reality, those intervals vary in complex ways inside the cell, and a single opacity value is not enough to describe them, which makes the problem difficult to solve.

4.2. Combined Solution. Because a single solution that would implement both water-to-body and body-to-water effects in all kinds of scenes in a satisfactory way is not yet available, we propose an alternative implementation that combines the strengths of the two approaches. Bodies are categorized according to their density to floaters and blockers. Floating bodies are advected with the flow and do not really need to block water. They can therefore be treated with the Thürey et al. method [11]. On the other hand, bodies that are denser than water do not float and can be treated with our method without the previously mentioned aliasing problems becoming an issue.

The Thürey et al. method only affects water in a single phase that changes the depth of the topmost water column. This can be implemented between steps 1 and 2 in our method without any adverse interaction with our method. Water-to-body effects in our examples are implemented as suggested above, using the drag and buoyancy forces, but also, for example, the triangle subdivision scheme of [13] could be used.

5. Results and Discussion

To evaluate and illustrate our results, we have created three interactive test scenes. The first two demonstrate blocking capabilities. In the first one, the user drops heavy cubes and two types of cars on a smooth hill with a large water source on top (Figures 8 and 9). The second demonstrates controlling

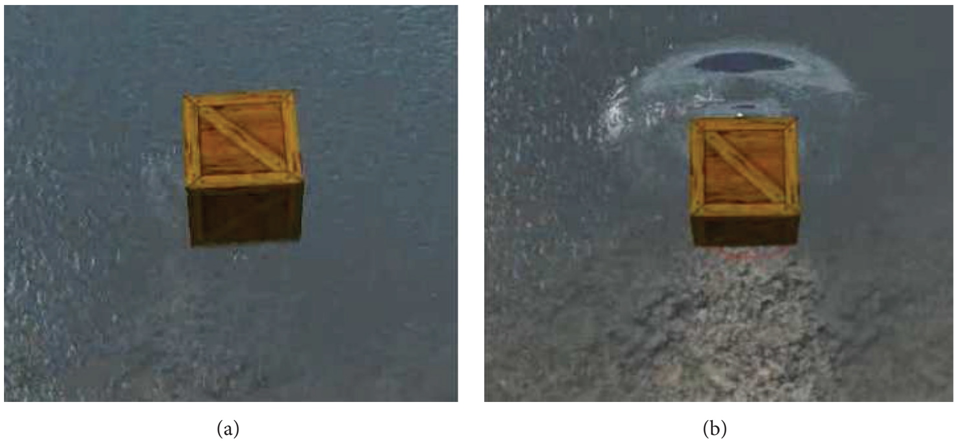
FIGURE 8: A visual comparison of the difference between a traditional [11] body (a) and a blocking body (our method, (b)) lying on a hill.

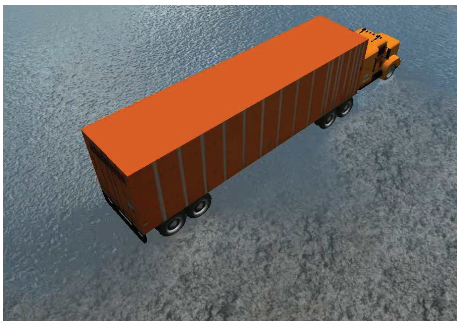
FIGURE 9: Since the vertical extents of blockers are taken into account, water can also flow below objects.

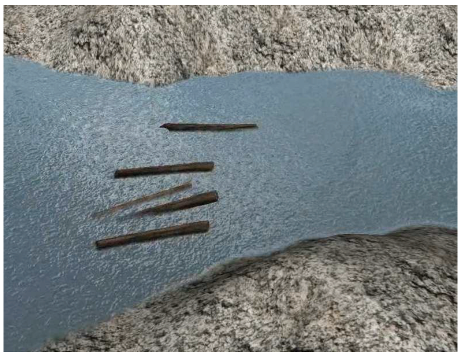
FIGURE 10: Logs floating on a river.

river flow by dropping blocking obstacles (Figures 11, 12, 13, and 14). The last one combines floating with blocking. It has logs floating down a river with a car crossing the river, hitting some of the logs on the way (Figures 1 and 10). The scenarios use either a 128 × 128 or a 256 × 256 simulation grid with a 1-meter simulation resolution. The time step is 20 ms.

We have elected to benchmark our method by comparing it to our implementation of the interaction method of [11], which we find to be the most advanced of the real-time heightfield methods, especially when it comes to handling heavy and large bodies that can be submerged. Other heightfield methods such as [9, 13, 14] would look very similar to the Thürey et al. method in our test cases with heavy objects, since none of these prevent water from passing through the bodies. For the floating objects, we do not claim that our method currently achieves a quality similar to any of the mentioned methods. A comparison to the significantly more advanced methods based on a 3D grid or particles is not included, since these cannot be directly applied to a heightfield-based simulation. None of the heightfield-based methods are currently even close to being competitive with them in quality, but, on the other hand, there is a large performance benefit when using heightfields.

The first two scenarios are handled well by our method. Water flows around the blocking rigid bodies and under a truck. Visual comparisons between our method and the one in [11] have been provided in Figure 8, Figures 11 through 13, and the accompanying video. The differences should be very apparent, since the Thürey et al. method [11] lets water mostly flow through the bodies, only creating limited interaction waves that are in many cases dwarfed by the existing water movement.

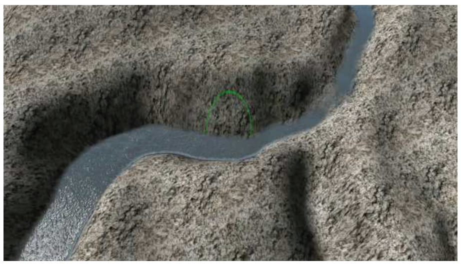
FIGURE 11: A free flowing river before building a dam.

The method of [15] also fares well in these cases. The main difference is in our pressure tracking, which allows for much nicer object/water borders than those achieved in [15].

In the third scenario, the logs can in principle be implemented as blocking bodies using our method. They float and are advected by the flow, but the spatial aliasing problem as

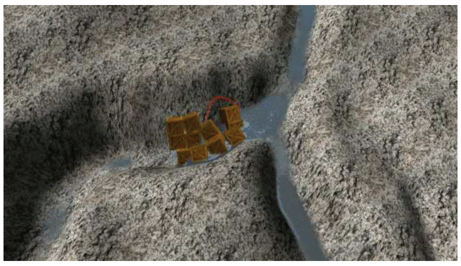
Figure 12: A dam built out of crates changes the river flow (our method).

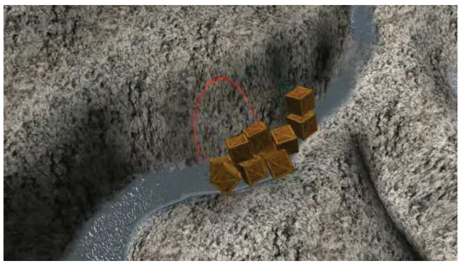
Figure 13: The same scene simulated using the method from [11], ($\alpha = .5$) does not alter the river flow much.

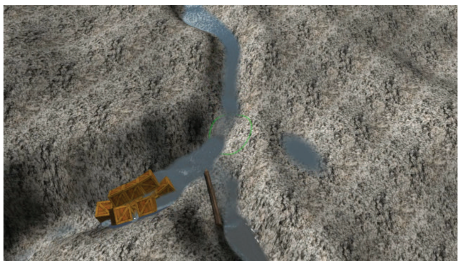
Figure 14: Combining our approach with the method of [11], blocking bodies can be used to interactively alter where floating bodies end up with the flow.

discussed in Section 4.1 makes the results unconvincing. This is one of the main limitations of our method. The advection is also somewhat limited, since the velocity under the body is usually zero. This could possibly be solved in the future by interpolating the velocity field for the effective surface similarly to pressure.

When the third scenario is implemented with the combined solution described in Section 4.2, the logs are classified as floaters and do not block water. The car is a blocker and is not affected by the water. We find this scenario to work much better with the combined solution than when implemented with either of the methods alone.

The stability of our method was tested empirically by varying the time step. It was found that, even with high-speed objects, the simulation is stable as long as the underlying pipe method itself is stable, that is, the rigid body interaction is not what limits the selection of the time step. The only direct effect our interaction method has on fluid velocities is decreasing them. The blocker update sometimes creates large height gradients when a cell is left empty, but we did not find any noticeable stability problems in practice even in those cases. In the combined solution, the first problem appearing when increasing the time step was the floating objects (whether they used our method or that of [11]) becoming unstable due to the torque caused by the buoyancy forces. Finally, when it comes to the aliasing problems, the time step does not play much of a role.

Our method and its predecessor [15] are the only heightfield-based interaction methods where flow is really blocked by the bodies. The other methods are concentrated on scenarios such as floating boats, where the body-to-water effect is mostly intended for visual, rather than interactive, purposes. However, we think that controlling water masses with rigid bodies will open up many kinds of new interaction possibilities in games and should be given more consideration. On the other hand, we also demonstrated that these two kinds of effects can coexist in a single scene.

The method of [15] also allows for this kind of interaction but is less grounded in physical simulation than our blocking solution. Most importantly, we take external pressure into account similarly to [29] and take steps toward a more general approach that would allow the method to be extended to any number of layers and more complex geometry. For example, new physical situations can now be simulated, such as the equalization of two columns of water that are connected by a tunnel, because the external pressure in the tunnel is included in the model.

A crucial addition compared to [15] in our method is the procedure for estimating the effective surface. The result is roughly the same surface that would exist in, for example, the method of [11] and can therefore be used for calculating water-to-body effects. It also has the necessary property of reacting smoothly to all kinds of geometry changes, which is necessary for retaining visual quality.

Thürey et al. [11] use a similar idea to our blocker handling step as the primary method for handling object-to-water effects. This has the problem that water is moved blindly to the surroundings, causing water to leak through a blocking body. In our approach, most of the work is done in a later step when calculating the new flow, which limits flow into the bodies.

In all examples, the water simulation and body-to-water effects are implemented on the GPU. The model is designed for implementation without random write access, which allows using fast GPU texture memory without any need for thread synchronization or atomic operations. Our implementation packs the simulation state into four grid-sized textures with four 32-bit floating point channels each (and their duplicates for double buffering).

On a laptop with an NVIDIA Quadro 1000 M, a simulation step takes about 5 ms of the GPU time on a 512×512 grid and 2 ms on a 256×256 grid. For the 1 m resolution grid in our examples, a 20 ms time step is stable enough. This makes the method feasible for real-time applications with large bodies of water, even with the limited GPU budget that this kind of system would be allowed in a game.

Solving the aliasing problems could result in a single method usable for all kinds of scenarios. A priority for future work is to either generalize the 2D approach of [10] to our more complicated case or modify and simplify a 3D approach such as the one in [37] to become compatible with our underlying 2D simulation and fast enough for large grids. Our method is also admittedly not rigorously based on physics. Momentum conservation is not considered, the solid-fluid velocity equality constraint is not enforced, and the depth update still has some ad hoc steps to improve stability. Finding a way to apply a completely physically based solution without losing too much speed is another important topic for future work. Our combined solution could also be improved by dynamically changing between the methods depending on the situation. For example, if a floating piece of ice gets stuck, it could be converted to a blocking body, allowing for a dam to be formed.

Our implementation concentrates on the water behavior and interaction and is visually fairly simple. Implementing adaptive tessellation, spray, foam, bubbles, breaking waves, and other effects using, for example, a particle system would naturally improve the visual quality of the results considerably. It could also be possible to extend the method to create, for example, foam based on the actual interaction, instead of the typical texture-based procedural methods used in current games (similarly to [42]).

6. Conclusions

We presented a GPU-parallel real-time water-solid interaction method for heightfield water simulation. Our method extends and generalizes that presented in [15]. It is based on bodies pushing down the water surface but includes a procedure for estimating where the surface would be without the body. This allows richer rigid body interaction in heightfield water simulation than what could be achieved using any of the previous methods.

A limitation of our method is that some aliasing artifacts are encountered especially when the method is used to handle floating objects. Despite trying various approaches, the artifacts could not be removed. To circumvent the problem, combining different methods in the same scene was proposed and demonstrated.

One of the most important remaining problems is to find a solution for the spatial aliasing problems inherent in the presented flow blocking method. This could allow a single method to be used for all kinds of bodies. Further physical realism should also be strived for, but it is important to do this without sacrificing performance. Various visual enhancements are also possible, including foam, and are integrating a particle system to create splashes.

The presented method is fast enough for real-time games on current systems, even with the need to dedicate most of the computational resources for various other game functionalities. The method can be combined with an off-the-shelf rigid body solver, enabling new kinds of player interaction with water.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was partly funded by KAUTE foundation. The author would also like to thank the editor and anonymous reviewers for their useful comments that helped to improve the quality of this paper.

References

[1] Where's My Water, 2014, http://www.edge-online.com/features/the-making-of-wheres-my-water.

[2] Sprinkle behind the scenes, 2014, http://tuxedolabs.blogspot.fi/2011/11/sprinkle-behind-scenes.html.

[3] "GDC Europe: Eric Chahi talks convergence of technology and design in project Dust," http://www.gamasutra.com/view/news/29953/GDC_Europe_Eric_Chahi_Talks_Convergence_Of_Technology_And_Design_In_Project_Dust.php.

[4] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 154–159, Eurographics Association, 2003.

[5] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Transactions on Graphics*, vol. 33, no. 4, p. 153, 2014.

[6] T. Kellomäki, "Water simulation methods for games: a comparison," in *Proceedings of the 16th International Academic MindTrek Conference 2012: "Envisioning Future Media Environments" (MindTrek '12)*, pp. 10–14, ACM, New York, NY, USA, October 2012.

[7] R. Bridson, *Fluid Simulation for Computer Graphics*, A K Peters/CRC Press, 2008.

[8] J. F. O'Brien and J. K. Hodgins, "Dynamic simulation of splashing fluids," in *Proceedings of the Computer Animation (CA '95)*, pp. 198–205, April 1995.

[9] M. Gomez, "Interactive simulation of water surfaces," *Game Programming Gems*, vol. 1, pp. 187–194, 2000.

[10] J. Tessendorf, "Interactive water surfaces," in *Game Programming Gems*, vol. 4, pp. 265–274, 2004.

[11] N. Thürey, M. Müller-Fischer, S. Schirm, and M. Gross, "Realtime breaking waves for shallow water simulations," in *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications (PG '07)*, pp. 39–46, 2007.

[12] C. Yuksel, D. H. House, and J. Keyser, "Wave particles," *ACM Transactions on Graphics*, vol. 26, no. 3, Article ID 1276501, 2007.

[13] N. Chentanez and M. Müller, "Real-time simulation of large bodies of water with small scale details," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposiumon Computer Animation (SCA '10)*, pp. 197–206, Eurographics Association, Aire-la-Ville, Switzerland, 2010.

[14] S. Liu and Y. Xiong, "Fast and stable simulation of virtual water scenes with interactions," *Virtual Reality*, vol. 17, no. 1, pp. 77–88, 2013.

[15] T. Kellomäki, "Interaction with dynamic large bodies in efficient, real-time water simulation," *Journal of WSCG*, vol. 21, no. 2, pp. 117–126, 2013.

[16] N. Foster and D. Metaxas, "Realistic animation of liquids," *Graphical Models and Image Processing*, vol. 58, no. 5, pp. 471–483, 1996.

[17] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner, "SPH fluids in computer graphics," in *Eurographics*

2014—State of the Art Reports, pp. 21–42, The Eurographics Association, 2014.

[18] M. Müller-Fischer, "Fast water simulation for games using height fields," in *Proceedings of the Game Developer's Conference*, 2008.

[19] S. Prescott, "Unreal engine 4 tech demo details particle and water effects," http://www.computerandvideogames.com/441374/unreal-engine-4-tech-demo-details-particle-and-water-effects/.

[20] J. J. Monaghan and J. B. Kajtar, "SPH particle boundary forces for arbitrary boundaries," *Computer Physics Communications*, vol. 180, no. 10, pp. 1811–1820, 2009.

[21] N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, and M. Teschner, "Versatile rigid-fluid coupling for incompressible SPH," *ACM Transactions on Graphics*, vol. 31, no. 4, article 62, 2012.

[22] N. Akinci, J. Cornelis, G. Akinci, and M. Teschner, "Coupling elastic solids with smoothed particle hydrodynamics fluids," *Computer Animation and Virtual Worlds*, vol. 24, no. 3-4, pp. 195–203, 2013.

[23] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw, "Efficient simulation of large bodies of water by coupling two and three dimensional techniques," *ACM Transactions on Graphics (TOG)*, vol. 25, no. 3, pp. 805–811, 2006.

[24] N. Chentanez and M. Müller, "Real-time Eulerian water simulation using a restricted tall cell grid," *ACM Transactions on Graphics (TOG)—Proceedings of ACM (SIGGRAPH '11)*, vol. 30, no. 4, article no. 82, 2011.

[25] W. Li, X. Wei, and A. Kaufman, "Implementing lattice Boltzmann computation on graphics hardware," *The Visual Computer*, vol. 19, no. 7-8, pp. 444–456, 2003.

[26] Y. Zhu and R. Bridson, "Animating sand as a fluid," *ACM Transactions on Graphics*, vol. 24, pp. 965–972, 2005.

[27] J. Tessendorf, "Simulating ocean water," SIGGRAPH Course Notes, 1999.

[28] M. Kass and G. Miller, "Rapid, stable fluid dynamics for computer graphics," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '90)*, pp. 49–57, ACM, New York, NY, USA, 1990.

[29] D. Mould and Y. H. Yang, "Modeling water for computer graphics," *Computers and Graphics*, vol. 21, no. 6, pp. 801–814, 1997.

[30] X. Mei, P. Decaudin, and B.-G. Hu, "Fast hydraulic erosion simulation and visualization on GPU," in *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications (PG '07)*, pp. 47–56, November 2007.

[31] O. Šťava, B. Beneš, M. Brisbin, and J. Křivánek, "Interactive terrain modeling using hydraulic erosion," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '08)*, pp. 201–210, Dublin, Ireland, 2008.

[32] M. M. Maes, T. Fujimoto, and N. Chiba, "Efficient animation of water flow on irregular terrains," in *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE '06)*, pp. 107–115, ACM, New York, NY, USA, December 2006.

[33] A. T. Layton and M. van de Panne, "A numerically efficient and stable algorithm for animating water waves," *The Visual Computer*, vol. 18, no. 1, pp. 41–53, 2002.

[34] T. Kellomäki and T. Saari, "A user study: is the advection step in shallow water equations really necessary?" in *Eurographics 2014-Short Papers*, pp. 41–44, The Eurographics Association, 2014.

[35] T. Takahashi, H. Ueki, A. Kunimatsu, and H. Fujii, "The simulation of fluid-rigid body interaction," in *Proceedings of the ACM SIGGRAPH Conference Abstracts and Applications (SIGGRAPH '02)*, p. 266, ACM, 2002.

[36] A. Robinson-Mosher, T. Shinar, J. Gretarsson, J. Su, and R. Fedkiw, "Two-way coupling of fluids to rigid and deformable solids and shells," *ACM Transactions on Graphics*, vol. 27, no. 3, article 46, 2008.

[37] C. Batty, F. Bertails, and R. Bridson, "A fast variational framework for accurate solid-fluid coupling," *ACM Transactions on Graphics*, vol. 26, no. 3, article 100, 2007.

[38] C. S. Peskin, "The immersed boundary method," *Acta Numerica*, vol. 11, pp. 479–517, 2002.

[39] N. Chentanez and M. Müller-Fischer, "A multigrid fluid pressure solver handling separating solid boundary conditions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 8, pp. 1191–1201, 2012.

[40] B. E. Feldman, J. F. O'Brien, and B. M. Klingner, "Animating gases with hybrid meshes," *ACM Transactions on Graphics*, vol. 24, pp. 904–909, 2005.

[41] E. Coumans, "Bullet physics library," http://bulletphysics.org/.

[42] N. Thürey, F. Sadlo, S. Schirm, M. Müller-Fischer, and M. Gross, "Real-time simulations of bubbles and foam within a shallow water framework," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, pp. 191–198, 2007.