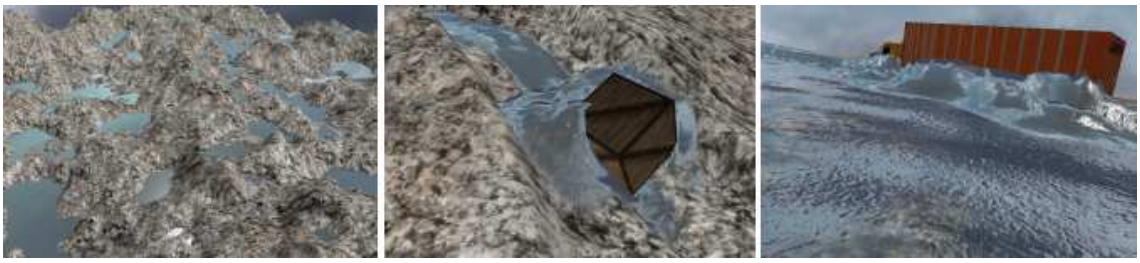
Interaction with Dynamic Large Bodies in Efficient, Real-Time Water Simulation

Timo Kellomäki
Tampere University of Technology
Department of Pervasive Computing
P.O.Box 553
FI-33101 Tampere
timo.kellomaki@tut.fi



Left: Natural lakes form on uneven terrain. Center, right: Examples of bodies blocking flow.

ABSTRACT

Water is an important part of nature. Interactively simulating large areas of flowing water would be a welcome addition to many virtual worlds, but the simulation is computationally demanding. Another problem is combining the simulation with rigid bodies, which are the most common interaction solution in virtual worlds. Heightfield water simulation is fast, but is especially hard to couple with rigid bodies: Usually water simply flows through the bodies. We propose a method that generalizes the extremely fast virtual pipe method to handle large, dynamic bodies. Our method diverts water around the objects. This enables us, for example, to dynamically build and destroy dams on rivers in a large virtual world.

Keywords

computer animation, games, rigid bodies, virtual worlds, water simulation

1 INTRODUCTION

Water is a fascinating element because it exhibits such complex behaviour. It is also always present in many forms in our daily lives. In video games and other real-time virtual worlds, rivers, lakes, and oceans are often implemented as static geometry with a variety of procedural techniques to improve the visual quality. As hardware improves, the trend is towards large and interactive worlds, where everything is allowed to change. The traditional approach is not always versatile enough in these situations. In many applications, the water flow will need to be simulated to see how streams behave and where lakes form.

There are several approaches to water simulation. Full 3D engineering methods can be extremely accurate and can capture the complex behavior of water down to the smallest detail. However, they are also extremely taxing on computational resources and often very complex. Unfortunately, they rarely run in real-time unless the simulation resolution is very small.

Luckily, virtual environments often only need to be convincing instead of realistic. Many of the details can be filled in using visual tricks. On the other hand, real-time performance is needed, and often even more, because the limited computational resources need to be split between several competing subsystems. Besides just the near-photorealistic graphics with a plethora of effects, also artificial intelligence and other such tasks need their share. If a water simulation system is to become popular in these applications, it needs to run in real-time with only a few percent of the available resources and be extremely easy to adopt.

There are many ways to simplify the traditional methods. One of the most common is to resort to heightfields, which reduce the number of cells needed from $O(n^3)$ to $O(n^2)$, where n is the number of cells per dimension (which determines the resolution). This means that even if hardware and algorithms progress to a point where full 3D large-scale simulation is viable in real time, heightfield methods will be able to simulate even larger areas (or with better resolution, whichever

is needed). Heightfield methods are unable to represent many interesting water phenomena, such as waterfalls or splashes. However, some of the shortcomings can be compensated by adding a particle system [OH95, HW04, CM10].

Our motivation is that, in our experience, even the simplest methods seem to be too much of a burden on resources for large-scale adoption, but the lack of realism does not pose such a problem. Therefore, we have chosen to build on the simplest possible model, based on virtual pipes.

The dynamics of virtual worlds is nowadays routinely based on off-the-shelf physics engines, which typically simulate rigid bodies. The bodies should naturally also interact with any simulated water that is present in the scene. The two simulations need to be coupled in two ways: Water affects the bodies via buoyancy, advection, and drag. On the other hand, the bodies affect the water, since water cannot enter them. This gives rise to waves and redirects the flow around the bodies.

Heightfield water simulation methods often implement an uneven terrain as another heightfield. Interaction with the terrain is well-established and easily implemented in many methods [MDH07, TMFSG07, CM10, Kel12]. However, two-way coupling with general, dynamic rigid bodies is more difficult. Water effects on objects are often easily handled, but object-to-water coupling causes problems. In all implementations we have seen, water enters the bodies and flows through them. Usually loosely physically based waves are generated around the object. This approach works when a small object is floating on a calm water surface, but makes no sense if the objects are large or the water flow is very dynamic (imagine floodwater from a dam break).

We present a novel coupling method that redirects the flow to avoid obstacles naturally. Water never enters the bodies. The method enables, e.g., building dams out of dynamic bodies. We focus on interaction with large and heavy objects that can block the flow. Therefore, we currently only implement the object to water coupling. That is, the objects do not float and are not pushed by the water. We do have plans to extend the model to cover these effects.

There is no perfect solution for the coupling, if the water is represented as a single heightfield. The objects may divide the water to arbitrarily many layers in the vertical direction and cause air pockets. We chose to simplify the situation by assuming that the bodies occupying each cell are vertically continuous and that there are no air pockets below the water surface. We ignore bodies that are above the water and do therefore not interact with it.

Our method is a generalization of the fast virtual pipe method, and runs easily in real-time for large grids using parallel computation enabled by the modern GPUs.

This paper is structured as follows. Section 2 recaps the relevant previous literature. The pipe method is described in Section 3. Our novel coupling algorithm is presented in Section 4. Section 5 evaluates the present work and establishes directions for future work.

2 RELATED WORK

In real-time animation, more or less procedural methods are often used to create waves on water. Examples of these methods are those based on the Fourier Transformation [Tes99]. These methods often create visually spectacular results for very large areas. However, they are mostly limited to visual effects on static areas of water. Dynamic flow over uneven terrain or rich interaction with moving rigid bodies does not fit naturally to these approaches.

Fluid simulation, on the other hand, has long traditions in the engineering discipline and is most often based on solving the *Navier–Stokes equations* (NSE) numerically. While this approach yields realistic results, such methods are very computation-intensive. These solvers can be divided into two categories. Eulerian methods subdivide the domain into a grid where the fluid properties are observed. Lagrangian methods track discrete particles as they are advected along the flow.

Since the full 3D methods are out of scope for this work, we refer the interested reader to Bridson [BMF07] on Eulerian methods and Solenthaler and Pajarola [SP09] on Lagrangian methods.

One hybrid method that is relevant to us is the tall-cell method, where a 3D simulation is employed near the surface, but a 2D simulation is used for deeper parts. An impressive recent example using an adaptive grid is presented in [CM11]. We especially find this method very promising for the future, since it only needs $O(n^2)$ cells just as heightfields, but still has many of the advantages of full 3D methods. However, the algorithms are very complex and the performance is still orders of magnitude slower than some of the 2D methods for a given horizontal resolution [Kel12].

2.1 Heightfield Methods

Almost all practical solutions still resort to heightfields, and the rest of this paper concentrates on those. They are naturally suited for large masses of water, such as rivers and oceans, where most of the water is relatively calm, since only the water surface is simulated.

Additional requirements for a heightfield water simulation method are dynamic free surfaces and interaction with a heightfield terrain. Free surfaces are needed if

the part of the domain that is occupied by water is allowed to change. Heightfield terrains are widely used in applications and allow for much more interesting situations than mere open-water simulations.

The *Shallow Water Equations* (SWE) are a simplification of the NSE, and can be solved to create a fast, yet realistic heightfield water simulation [LvdP02]. Recently, Chentanez and Müller added breaking waves and made many other improvements, yielding a fast method suitable for large areas [CM10].

An approach based on wave particles was proposed in [YHK07], and some researchers use Lattice Boltzmann methods [LWK03, GCTW10]. Both of these methods are well suited for open waters, but to our knowledge, have not been extended to cope with uneven terrain and dynamic free surfaces.

The simplest heightfield method is based on modeling *virtual pipes* between columns. This method was pioneered by Kass and Miller [KM90], who simplified the SWE further to reach the 2D wave equation on the water surface. They already note that the results are not realistic enough for engineering, but are promising for visualization purposes. O'Brien and Hodgins [OH95] extended the method by adding particles to overcome the limits of heightfields and used an intuitively simple virtual pipe formulation, which we adopt in this paper.

The model has then been developed further. Mould and Yang [MY97] added a heightfield terrain below the water and generalized the pipe model to consist of several layers per column, which adds realism by removing the assumption of vertical isotropy. Holmberg and Wünsche [HW04] combined the model with particles to simulate natural phenomena such as rivers and waterfalls. Maes *et al.* [MFC06] implemented the method on the GPU.

The pipe method is naturally parallel and therefore straightforward to implement efficiently on the GPU. The method is extremely fast and simple, yet captures the main dynamics of water flowing on irregular terrains. The most obvious downside is the lack of vortices, which create much of the interesting behavior of water. However, even this simple method would be a huge improvement to many current virtual worlds that completely lack dynamic water.

Even more interesting scenarios can be simulated if the heightfield terrain is allowed to be dynamic. All of these problems have been solved for the pipe method (e.g. [MDH07]) and SWE (e.g. [CM10]. For an impressive example of these techniques, see the Augmented Reality Sandbox [1]. However, since the terrain is still always a heightfield, even more dynamic situations can be achieved using generic rigid bodies.

[1] `http://idav.ucdavis.edu/~okreylos/ResDev/SARndbox/` (cited March 8, 2013)

2.2 Coupling with Rigid Bodies

There exists a large literature of sophisticated solid-fluid coupling methods for full 3D methods. See, e.g., [AIA+12] for a recent solution in the Lagrangian framework and [RMEF09] for an Eulerian approach. However, the 3D methods are still rarely real-time, so they typically aim for much more realistic solutions than is affordable in our context of large-scale real-time simulation. Therefore, we concentrate on solutions applicable to heightfield-based fluids.

The effects of water on rigid bodies are typically easy to model in all of the heightfield methods and many publications include some kind of a solution [YHK07, CM10, OH95, MY97, TMFSG07]. The body-to-water effect is more difficult, since even a single, convex body violates the heightfield assumption when submerged. It is obvious that there can be no physically based solution without generalising the heightfield somehow, e.g., to consist of more than one layer.

Previous solutions concentrate on small, floating objects or objects that are dropped into the water. Various methods are used to generate waves. In practically all methods we are aware of, water is allowed to enter the bodies. This does solve the problem of objects breaking the heightfield assumption, but is not believable unless the object is very small.

O'Brien and Hodgins [OH95] discuss the case of an object hitting the surface. A force is estimated so that the entering mass is situated nearly on the surface. This causes an external pressure to the water, causing it to flow to the neighboring columns. The model is physically based only for an object with no volume. A similar model is adopted by Mould and Yang [MY97].

Thürey *et al.* [TMFSG07] recognize the problem of the previous methods not taking volume into account. They first mention pushing the water columns down until overlaps are removed and redistributing the removed water in the neighborhood. As they state, this method would run into problems if the object is fully submerged, because the water does not close the gap above the object.

Thürey *et al.* citeThurey therefore propose a method where the volume of displaced water is tracked in each cell. The change in this volume is distributed to neighboring cells and can be negative, closing up the gaps caused by submerged objects. This corresponds closely to our virtual volume described in Section . In their method, water still enters the volume of the body and flows through it.

To our knowledge, our approach is the first heightfield-based method where water does not enter the bodies. We see this as a compulsory first step towards more physically based coupling with rigid bodies.

3 THE PIPE MODEL

Our version of the pipe model is mostly the same as in Mei *et al*. [MDH07], but the model is briefly repeated here to help the reader. We have made only minor changes and additions.

The scene is divided to a uniform grid of square cells. The variables that are tracked at each cell center (x,y) are terrain height $h(x,y)$ and depth of water $d(x,y)$. We denote the total water level by $H(x,y) = h(x,y) + d(x,y)$. We will often omit the cell coordinates whenever there is no danger of confusion.

Each cell is connected to its four direct neighbors (the von Neumann neighborhood) by a virtual pipe. Some versions use the Moore neighborhood, but since we are aiming for an extremely lightweight method, we do not find doubling the calculations worth the added quality.

The pipes store current outflow $f_o(x,y,i)$, where $i \in \{L,R,U,D\}$ is the direction from the cell (x,y). The outflows from a cell form the vector $f_o = [f_o(L), f_o(R), f_o(U), f_o(D)]$.

To update from time t to $t+1$, any water from external sources is first added to the depth. Then, the new flow is created by the pressure difference at the heads of each pipe. The created flow is also proportional to the cross-section of the pipe, but neither O'Brien and Hodgins [OH95] or Mei *et al* [MDH07] explicitly mention how they calculate it: Is it just an assumed constant or is the geometry of the situation taken into account?

Our solution, inspired by the cross-sections used in [MY97], is to calculate the height of the interval $I(x,y,i)$ where the water column at (x,y) touches air in the direction i, e.g., $|I(x,y,L)| = \max(h(x,y), H(x-1,y)) - H(x,y)$. The result is proportional to the pressure difference, but takes into account the case where the water level at the target cell is lower than the terrain at the source. In addition, we use a constant pipe area A that can be set by the user to change water behavior. Now, new flow is created in proportion to the height of the water–air interval in all four directions:

$$f_o^{t+1}(x,y,i) := \max(0, \mu f_o^t(x,y,i) + \Delta t \cdot A \frac{g|I(x,y,i)|}{\ell}), \quad (1)$$

where $i \in \{L,R,U,D\}$ is the direction, μ is an artificial friction coefficient as in [MY97], Δt is the time step, A is the pipe area (constant for all pipes), g is the acceleration due to gravity, and ℓ is the pipe length. $|I(x,y,i)|$ is the interval height as discussed above.

To prevent negative water levels from occurring, a factor K is calculated in each cell. It is chosen so that scaling by it limits the total outflow to the amount of water in the cell, as in [MDH07]:

$$K = \min(1, \frac{d}{\Delta t \sum f_o}), \quad (2)$$

where the sum is over the four elements of f_o and thus equals the total outflow.

Finally, the water depth is updated. For this, an inflow vector $f_i(x,y)$ analogous to the outflow vector is gathered from the scaled outflows at the corresponding neighbors. For example, $f_i(x,y,L) = K(x-1,y) \cdot f_o(x-1,y,R)$. Now, the final change of depth is calculated simply as follows:

$$\Delta d = \frac{\Delta t \cdot (\sum f_i - K \cdot \sum f_o)}{\ell^2}. \quad (3)$$

The horizontal velocity field $v(x,y)$ is needed for visual effects (and possibly water-to-object coupling in the future) and is calculated almost as in [MDH07]. For the left-right direction

$$v_x = \frac{f_i(L) - f_o(L) + f_o(R) - f_i(R)}{2\ell d}. \quad (4)$$

The calculation is similar for the up-down direction.

We handle the domain boundaries by not running the simulation on the border cells. The border cell depths are initialized to zero and never updated. This is not a realistic solution, but in the pipe method this is enough to make the borders drain water with little reflection. Other conditions could be used if needed, but this simple solution suits our purposes.

To achieve stability, a smaller timestep is required as ℓ is decreased [MDH07]. With some parameter balancing, we did not encounter any stability problems with timesteps up to $\Delta t = 25$ ms and $\ell = 1$ m. See, e.g., [MDH07] for further discussion.

4 COUPLING WITH RIGID BODIES

We now present our novel coupling method. It is a generalization of the pipe method described above in the sense that if no bodies are present, everything reduces to the basic method.

We aim for a situation where the bodies block flow, which creates most of the coupling physically. Therefore, the basic invariant in our model is that water can never exist inside a rigid body (to the precision of the simulation grid). We take that this is a necessary assumption for rich and believable water–body interaction, but it has been neglected in the previous height-field methods.

Our method assumes that each cell is blocked by a single contiguous vertical interval of bodies, $[b_-, b_+]$, where b_- is the lower limit of the interval and b_+ the upper. It is also useful to ignore bodies that are completely above the water. We assume that $b_- = \infty$ and $b_+ = -\infty$ if no body is present in a cell.

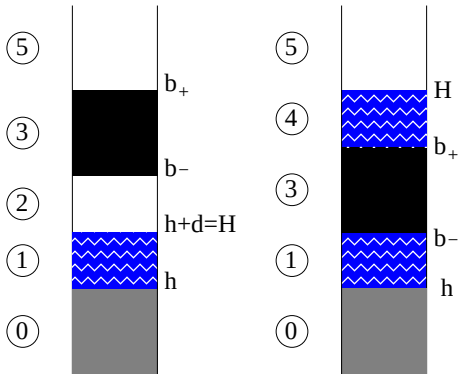


Figure 1: The layers in a single cell. Left: no water above the object (object is floating if ② is empty). Right: water above the object. In this case, water level H is calculated as terrain level + amount of water + height of body.

The method could be extended to have a number of layers similarly to what Mould and Yang [MY97] use to get rid of vertical velocity isotropy. However, the performance cost would probably outweigh the benefits for most applications. Storing several layers would make it necessary to store and update multiple depth and flow values per cell or pipe even in areas without bodies. Also drawing the result would become more complex. We therefore stay with a single-layer model.

There are two fundamentally different situations, which are differentiated by whether there is water above the blocking body. To simplify the terminology, we call these ”floating” (no water on the body) and ”submerged”, even if the floating state also includes situations where the body is not touching the water or has just hit the surface but is sinking rapidly (which is always the case in the current version of the method, since there is no buoyancy yet).

The layers in a single cell are presented in Fig. 1, where the contents are divided into six disjoint and possibly empty intervals in the vertical direction. The intervals are from bottom up: ⓪ terrain, ① lower water layer, ② air below body, ③ body, ④ upper water layer and ⑤ air above the body. Terrain continues to $-\infty$ and air to ∞. If no body is present, intervals ③–⑤ are empty (so of the two air layers, ② is chosen to exist). $|\cdot|$ denotes the height of an interval.

To reduce the storage and processing needs, we make another assumption. If there is water above a body, there is no air below it. The body in each cell is always either completely above the water, touching the water, or submerged. This means that either ② or ④ must be empty. This way we only need to store a single value for the amount of water per cell, and a single flow value per pipe. More importantly, blocking the water from entering the body becomes easier. Another benefit is that drawing the water remains simple, since a single heightfield is easy to build. On the other hand, our method erroneously blocks flow in situations such as b) in Fig. 2.

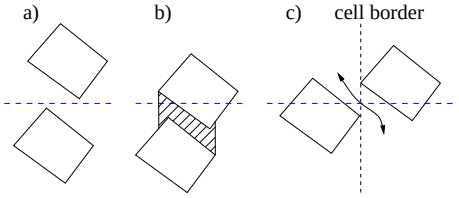


Figure 2: Some situations with multiple bodies. In a), the upper body is ignored since it does not touch water. In b), the area between bodies is erroneously blocked because of our assumption of contiguous blockers in the vertical direction. In c), two bodies are in neighboring cells and water should flow between them, but another assumption blocks this (see 4.3 for discussion).

We modify the depths described previously so that d describes the total amount of water in the cell. Some of it might be below the bodies and some above ($d = |①| + |④|$). We generalize H to mean the water surface level, taking possible bodies in the cell into account:

$$H = h + d + \begin{cases} 0 & \text{if } h + d \leq b_-, \\ b_+ - b_- & \text{otherwise.} \end{cases} \quad (5)$$

It is essential to understand that we only store h and d directly. The water level H is calculated when needed. If a body is floating, $H = h + d$, but for submerged bodies the two differ. This is demonstrated by the dashed line in Fig. 3.

For each time step, our method consists of the following phases: (a) simulate rigid bodies, (b) calculate blocked interval, (c) fix invariant and conserve volume, (d) calculate flow, (e) update depth, (f) prepare heightfield, and finally, (g) draw. The phases are described in detail below.

Phases b–f are implemented as OpenGL fragment shader passes, so each cell is processed in parallel. The method is crafted to make the cells independent and to only use information from the previous phases.

4.1 Finding Blocked Intervals (Phase b)

To find the blocked interval $b = [b_-, b_+]$ at each cell, the rigid body geometry is rendered from above and below into a texture using orthographic projection. The texture has the size of our simulation domain. Thus each texel corresponds to a single cell. The first pass writes the upper limits b_+ to a single channel of the texture using the depth buffer. The second pass inverts the depth test and writes the lower limits b_- to a second channel. The bodies that are completely above water level are not drawn.

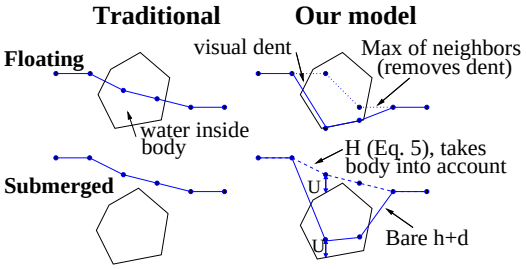


Figure 3: Comparing methods in different situations. Left: traditional solution, where water penetrates the body. Right: our method. Above: floating body, below: submerged body. Dots represent values in the cell centers. The dashed line shows calculated H, which is different from $h+d$ for submerged bodies (U is the height of water above the body). Dotted line shows the visual treatment for dents near floating bodies (see Subsection 4.4 and Fig. 7).

4.2 Invariant Update and Volume Conservation (Phase c)

After rigid bodies have moved, water may be located inside the bodies, which breaks our invariant. We need to fix this while conserving volume. This proves more difficult than one would imagine, since the blocked vertical interval can vary greatly due to aliasing caused by the grid as the body rotates. A straightforward idea would be to simply hold the amount of water above the object constant, but we found this solution unworkable because of the instability it causes. Instead, we solve this phase in a novel way that evades the instability. We first remove the old body and then add the new one. For this, we need the blocked interval at previous and current timesteps, b^t and b^{t+1}.

In this phase, we often need to add or remove water. To hold volume constant, each cell tracks *virtual volume* V: positive if water was removed and needs to be reinserted in the vicinity; negative if water was added and needs to be removed. A constant fraction τ of this volume is spread at the beginning of this step to each of the four neighbouring cells. Some volume loss could be caused if negative virtual volume ends up in areas where no water exists.

First the old body is removed. If b^t is non-empty and $b^t_+ < h+d$, the body in this cell was submerged and the volume underwater was $U_1 = b^t_+ - b^t_-$. In this case we add U_1 to the depth, holding the water surface H constant. Volume is conserved in the long run by subtracting U_1 from V, possibly causing negative virtual volume. If the body was floating, $U_1 = 0$ and nothing needs to be done.

We then insert the new body, removing any water from the interval it occupies. If $b^{t+1}_- < h+d$, the new body is underwater and the submerged volume $U_2 = \min(h+d-b^{t+1}_-, |b^{t+1}|)$. U_2 is removed from depth d and correspondingly added to V, again conserving volume. If the new body is floating, this pushes H down to b_-. Otherwise, the body is above the water, $U_2 = 0$ and H stays constant.

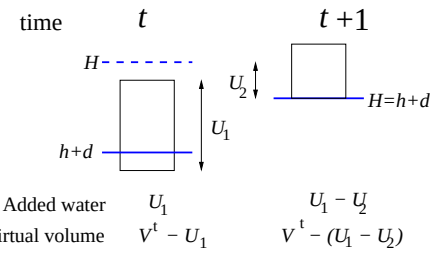


Figure 4: An example of enforcing the invariant. The submerged body at time t is removed and water amounting to U_1 is added so that H stays unchanged. In this case, the body at time $t+1$ is floating (original H is inside it), so H needs to be pushed down by U_2. The volume changes are subtracted from the original virtual volume V^t to conserve water volume.

Finally, if there is currently no body in the cell, a portion $\alpha \in (0,1]$ of the virtual volume is converted to actual water. If the virtual water is negative, water is removed, but naturally only until there is no water in the cell.

An example of removing and adding a body in this step is seen in Fig. 4. In this case, $U_1 > U_2$ so more water is added than removed. This causes some negative virtual volume, which will spread to the nearby cells and destroy a corresponding amount of water.

Virtual volume is inspired by Thürey *et al.* [TMFSG07]. They use a functionally very similar idea to remove some water from inside the bodies and propagate it to the neighboring cells, which is the core of their object-to-water coupling. However, we found that in both their and our method, the constant α needs to be rather small in order to keep the long time steps stable (we use $\alpha = 0.01$ for $\delta t = 20$ms). This means that the waves caused by the objects are also small and something more is needed.

4.3 Calculating Blocked Flow (Phase d)

Flow is governed by Eq. 1. We need to block two parts that are represented by the two terms on the right side of the equation: the existing flow (μf^t_o) and the new flow.

Let us first tackle the new flow, which is proportional to $|I|$, the height of the air–water interval. Consider the outflow from cell i to its neighbor j. The outflow can be caused by the water either above or below the object in cell i, i.e., intervals ①$_i$ or ④$_i$. The air part could be either of the two air intervals of cell j, i.e., ②$_j$ or ⑤$_j$. The intersections of these intervals are what cause new flow. The total length of water–air interface from i to j is therefore

$$|I'| = |①_i \cap (②_j \cup ⑤_j)| + |④_i \cap (②_j \cup ⑤_j)|. \quad (6)$$

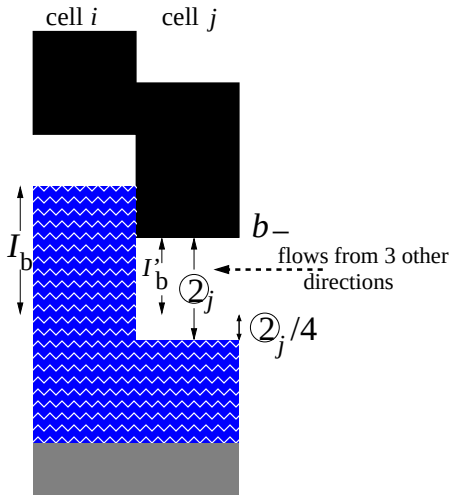


Figure 5: Limiting the existing flow below a body. I_b is the unlimited interval that is about to flow out during a time step. Limiting it by the lower border of the body in the neighbor, b_-, results in I'_b. But the binding limit here is caused by requiring that $②_j$ is not overfilled ($|I'_b| \leq |②_j|/4$, since flows come from four directions).

Recall that if no bodies are present, intervals ④ and ⑤ are empty and the formula reduces to that of the basic method.

The result of Eq. 6 is used in Eq. 1 instead of $|I|$ to calculate the new outflow vector f_o. This takes care of blocking the new flow.

Let us now handle the existing flow. We find the interval I from where water is leaving. The amount of water in the cell is now d^t and would be $d^{t+1} = d^t - \Delta t \sum f_o$ after the outflow. From d^t and d^{t+1} we calculate I using Eq. 5 with both values d^t and d^{t+1}. I is divided to parts leaving from above and below the body, I_a and I_b. If a body is present, one of them is almost always zero. If no body is present, we set $I_a = I_b = I$.

First of all, we only take into account flows from above to above and from below to below. In practice, the other flows are very rare: They can only happen when two distinct objects are right next to each other as in case c) of Fig. 2. A small change in positions would reduce this to case b) where no flow occurs anyway. Therefore this extra assumption is rarely relevant.

Now, I_a is limited from below by b_+ in the target neighbor and I_b similarly from above by b_-. The existing flows are limited in proportion to the intervals (i.e., if half of the interval is blocked, only half of the unblocked flow is allowed). Note that if no body is present in the neighbor, $b_+ = -\infty$ and $b_- = \infty$, and the flow is not limited in that case.

However, there is a complication. The interval $②_j$ at the target cell must not be overfilled, because that would cause water to enter the body. But we cannot know the inflows from the other directions without breaking the parallel structure or adding an additional iterative phase to the process. We overcome this problem with the following approximate solution.

Sum of the inflows below any body at the target cell j must be limited to $|②_j|$. Since there are four neighbours, we set a maximum of $|②_j/4|$ to each I_b going to cell j and limit the flow accordingly. This prevents $②_j$ from overfilling, but it is still filled eventually. This solution could cause artifacts in some situations, e.g., when water is flowing fast just below a bridge. However, in our experience it is an essential addition, since the lack of this limitation is easily visible, but the problems caused by it are not. The process of limiting I_b is visualized in Fig. 5.

The limited outflow from the cell is now $\max(I_a, I_b)/\Delta t$, i.e., we take the larger of the two intervals and convert it back to flow. This works both when no body is present and when either I_a or I_b is zero. The flow is further limited by K according to Equation 2, just as in the basic simulation.

A somewhat problematic situation that is unaddressed by this method happens when water flows from above a body but there is room for it below the body in the neighbor j, i.e., $|I_a| > 0$ and $|②_j| > 0$. In this case, water is transferred through the body from above to below. However, we have not noticed this visually in our tests.

4.4 Updating Depth and Constructing the Heightfield (Phases e and f)

Now that the flows have been limited, depth can be updated as in the basic method using Eq. 3. The velocity field is also calculated just as before using Eq. 4.

Before drawing the water, we need an extra phase to build the heightfield. The simplest implementation is to calculate H from the stored variables h, d, b_-, and b_+ using Eq. 5, which is indicated by the dashed line in Fig. 3. This approach causes visual dents near floating objects, because the object pushes water down below itself and thus the water mesh seems to curve down already outside the object.

We treat the denting problem by finding the cells where the object has probably pushed water down (b_- is within some ε of H). We adjust the values at these cells to be the maximum of H values in their Moore neighborhood. This value will most probably be the level from outside the body and takes care of most of the visual problems. This is demonstrated by the dotted line in Fig. 3.

Some problems still remain, however. Objects that are nearly submerged sometimes cause flicker, because small changes in d can cause large variations in H. The water-object borders are not always as clean as they could be.

Figure 6: A dam is built on a river.

Luckily, many of these problems could be masked by adding foam and particle effects to places where rapid change is happening, since the problems occur exactly where the foam would normally appear. This is an important task for future work.

5 EVALUATION AND FUTURE WORK

We have implemented the proposed method using OpenGL and GLSL shaders. The implementation uses relatively basic rendering techniques. One of the most important visual feature is visualizing the flow by using two bump textures with alternating weights. The velocity field generated by the method is used to advect these textures. The textures are reset when their weight is zero to prevent excess stretching. This method is used in many current games, e.g., [Vla10].

The rigid bodies are simulated using the open source Bullet physics engine[2]. The engine handles body dynamics and collisions. Currently the physics simulation is done on the CPU. The bounding boxes of the bodies are needed when eliminating bodies that are above water level in Section 4.3. We read back the H values and do the elimination on the CPU. This is a major performance problem that can be avoided in the future by simulating the bodies on the GPU and sharing data between the two simulations.

Our method prevents water from entering and flowing through the bodies. Water also visually seems to flow around the objects due to the calculated velocity field.

[2] `http://bulletphysics.com/`

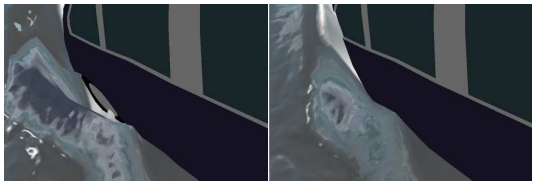


Figure 7: An example of our fix for visual dents near objects. Left: unfixed heighfield curves down already outside the object. Right: fixed border.

In addition, our method enables building dams out of dynamic rigid bodies. None of these come naturally with the previous methods that allow water to enter the bodies.

For comparison, we have implemented the rigid body coupling method suggested in Thürey *et al*. [TMFSG07]. Their method is best suited for floating objects, but also supports submerged bodies and is the state of the art in heightfield water physics coupling as far as we know. The accompanying video contains a scene where a wave reaches a large and heavy dynamic object, either passing through it almost unaffected (Thürey *et al*.) or actually going around it (our method). The video also demonstrates how our method creates a correct velocity field, which flows around the body instead of into it.

Our simulation takes approximately 18 ms on a grid of 1024×1024 on our test laptop with NVIDIA Quadro 1000M. Since our time step is 25 ms, about 1.4M cells can be simulated in real-time. More relevant results for applications are 4.7 ms on a 512×512 grid (5.3 times real-time) and 1.6 ms on a 256×256 grid (16 times real-time). Performance is currently limited by texture bandwidth, because several of the phases need a lot of information about their neighbors.

In a recent study, Kellomäki [Kel12] compares the performance of some methods. Although these performance comparisons are very rough, we can conclude that our generalized pipe method is a few times slower than the original pipe method, but still roughly an order of magnitude faster than the other methods surveyed.

The performance of our implementation greatly suffers from the fact that we have currently implemented only a single version of the simulation shaders, which are relatively complex. An obvious optimization would be to first simulate the whole scene using the simple and fast basic method, and then recalculate the comparatively small areas that are covered by the bounding boxes of rigid bodies, using the complex shaders. This would probably reduce the time used tremendously at the cost of a few extra draw calls and pipeline state changes. On the other hand, our current implementation is not much slowed by additional bodies.

One of the largest problems in our current implementation is caused by the fact that the water surface does

Figure 8: A box is dropped into a calm lake.

not just pass through the objects anymore. This causes visual problems like dents around the bodies and flickering, some of which we have already addressed, but some of which still remain. The visual problems are apparent in the accompanying video, especially when compared to the clean borders achieved by the traditional method. However, we believe further work can solve these problems.

Traditional methods for calculating water-to-body coupling cannot be used with our solution. For example, buoyancy is traditionally proportional to the difference of water level and the lower limit of the body. In our method that information is not available, because water is pushed down by the object. Further work is needed here, e.g., to enable combining the physically based approach of O'Brien and Hodgins [OH95] with our method.

We also made some restrictive assumptions and sometimes resorted to ad hoc solutions to keep performance and stability good. Mainly, the bodies overlapping each cell need to be contiguous. Violating the assumptions becomes a problem less often than one might imagine, since most practical situations only have a single body per cell, and many objects are convex in the vertical direction. The assumptions need not be fully obeyed to achieve visually acceptable results.

One of the relevant problematic cases is a small hole in a dam. This could conceivably be overcome by treating bodies that are touching the terrain in each cell as temporary parts of the terrain itself. This should be contrasted to the previous methods, which only work correctly for objects with no volume and where no dams could be built at all (except from the terrain itself). Another limitation is that no air pockets are allowed under submerged bodies, but those would typically not be visible to the user anyway.

One of the possible next steps is to learn from the more sophisticated methods used in the 3D water simulation literature. Those methods need to be adapted to work in the heightfield context and also severely simplified, since they are still much too slow for application in actual real-time virtual environments. Another possibility is using the layered model of Mould and Yang [MY97] to separate the water above and below the objects.

Several other improvements are certainly possible. Most obviously, many additional visual techniques could be added to mask the simple simulation model used. We are currently not using any particle system to create splashes and other phenomena that cannot be represented by heightfields. We are also investigating the possibility of seamlessly converting all water to particles near the bodies and using full 3D simulation in those areas.

6 CONCLUSION

This work has described a novel method for handling object-to-water coupling in the context of heightfield water simulation. Although we elected to use the pipe method, the often-used shallow water simulations could also benefit from an approach where no flow is allowed through bodies.

We have built a prototype implementation that allows completely new kind of interaction in heightfield water simulation, such as dynamically building and breaking dams using rigid bodies. The result is very fast and, despite several assumptions, works acceptably in many practical situations.

We think that even an ad hoc method is better than the current situation, where there is simply no coupling solution at all available for large objects in large-scale heightfield water simulations. Furthermore, there is no fundamental reason why further work could not extend our flow blocking based method to also handle water-to-object coupling. On the other hand, all other existing water-to-object coupling methods are inherently not physically based because of their approach where water is allowed inside the bodies.

The problem field is becoming important since many, if not most, virtual environments rely heavily on rigid body physics for interaction and dynamics. Water that passes right through large dynamic objects is simply not believable enough in this context.

Furthermore, hardware has finally improved far enough to make the fastest water simulation methods viable in actual use cases, instead of being limited to mere research prototypes. We are on our way to having interactive water as just another easily added component in video games and other virtual environments.

7 REFERENCES

[AIA+12] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner, *Versatile rigid-fluid coupling for incompressible SPH*, ACM Transactions on Graphics (TOG) **31** (2012), no. 4, 62.

[BMF07] Robert Bridson and Matthias Müller-Fischer, *Fluid simulation: SIGGRAPH 2007 course notes*, ACM SIGGRAPH 2007 courses (New York, NY, USA), SIGGRAPH '07, ACM, 2007, pp. 1–81.

[CM10] Nuttapong Chentanez and Matthias Müller, *Real-time simulation of large bodies of water with small scale details*, Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Aire-la-Ville, Switzerland), SCA '10, Eurographics Association, 2010, pp. 197–206.

[CM11] Nuttapong Chentanez and Matthias Müller, *Real-time Eulerian water simulation using a restricted tall cell grid*, ACM SIGGRAPH 2011 papers (New York, NY, USA), SIGGRAPH '11, ACM, 2011, pp. 82:1–82:10.

[GCTW10] Robert Geist, Christopher Corsi, Jerry Tessendorf, and James Westall, *Lattice-boltzmann water waves*, Advances in Visual Computing (2010), 74–85.

[HW04] Nathan Holmberg and Burkhard C Wünsche, *Efficient modeling and rendering of turbulent water over natural terrain*, Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, ACM, 2004, pp. 15–22.

[Kel12] Timo Kellomäki, *Water simulation methods for games: a comparison*, Proceeding of the 16th International Academic MindTrek Conference (New York, NY, USA), MindTrek '12, ACM, 2012, pp. 10–14.

[KM90] Michael Kass and Gavin Miller, *Rapid, stable fluid dynamics for computer graphics*, Proceedings of the 17th annual conference on Computer graphics and interactive techniques (New York, NY, USA), SIGGRAPH '90, ACM, 1990, pp. 49–57.

[LvdP02] Anita Layton and Michiel van de Panne, *A numerically efficient and stable algorithm for animating water waves*, The Visual Computer **18** (2002), no. 1, 41–53.

[LWK03] Wei Li, Xiaoming Wei, and Arie Kaufman, *Implementing lattice boltzmann computation on graphics hardware*, The Visual Computer **19** (2003), no. 7, 444–456.

[MDH07] Xing Mei, Philippe Decaudin, and Bao-Gang Hu, *Fast hydraulic erosion simulation and visualization on GPU*, Computer Graphics and Applications, 2007. PG '07. 15th Pacific Conference on, Nov 2007, pp. 47–56.

[MFC06] Marcelo M. Maes, Tadahiro Fujimoto, and Norishige Chiba, *Efficient animation of water flow on irregular terrains*, Proc. of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia (New York, NY, USA), GRAPHITE '06, ACM, 2006, pp. 107–115.

[MY97] David Mould and Yee-Hong Yang, *Modeling water for computer graphics*, Computers and Graphics **21** (1997), no. 6, 801 – 814, Graphics in Electronic Printing and Publishing.

[OH95] James F. O'Brien and Jessica K. Hodgins, *Dynamic simulation of splashing fluids*, Computer Animation '95., Proceedings., Apr 1995, pp. 198–205.

[RMEF09] Avi Robinson-Mosher, R Elliot English, and Ronald Fedkiw, *Accurate tangential velocities for solid fluid coupling*, Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, ACM, 2009, pp. 227–236.

[SP09] Barbara Solenthaler and Renato Pajarola, *Predictive-corrective incompressible SPH*, ACM Trans. Graph. **28** (2009), no. 3, 40:1–40:6.

[Tes99] Jerry Tessendorf, *Simulating ocean water*, SIGGRAPH course notes, 1999.

[TMFSG07] Nils Thürey, Matthias Müller-Fischer, Simon Schirm, and Markus Gross, *Real-time breaking waves for shallow water simulations*, Proc. of 15th Pacific Conference on Computer Graphics and Applications, 2007, PG '07, 2007, pp. 39–46.

[Vla10] Alex Vlachos, *Water flow in Portal 2*, ACM SIGGRAPH 2010 courses (New York, NY, USA), SIGGRAPH '10, ACM, 2010, pp. 1–54.

[YHK07] Cem Yuksel, Donald H. House, and John Keyser, *Wave particles*, ACM Trans. Graph. **26** (2007), no. 3.