Fast Water Simulation Methods for Games

TIMO KELLOMÄKI, Colossal Order Ltd (formerly Tampere University of Technology)

Easy-to-use physics engines have created a whole new source of emergence and fun for digital games. Water simulation could add another similar emergent interaction element in 3D games. Several barriers that prevent this step for games with large playing areas are analysed. One of the most important problems is how to couple the water and physics simulations. Our implementation of the extremely fast virtual pipe method is compared with more sophisticated solvers. Also, two different implementations of physics coupling are compared.

CCS Concepts: • **Computing methodologies** → **Interactive simulation**; *Real-time simulation*; *Massively parallel and high-performance simulations*; *Physical simulation*; • **Applied computing** → **Computer games**; • **Human-centered computing** → *Interaction paradigms*; • **Software and its engineering** → Software performance;

Additional Key Words and Phrases: Water simulation, rigid body coupling, pipe method, games, realtime, fluid simulation

ACM Reference format:
Timo Kellomäki. 2017. Fast Water Simulation Methods for Games. *ACM Comput. Entertain.* 16, 1, Article 2 (December 2017), 14 pages.
https://doi.org/10.1145/2700533

1 INTRODUCTION

It can be argued that much of the fun of playing digital games comes from the quality of the core interaction with the game world. An emergent system is one that can create complex behaviour from simple parts. It is therefore easy to learn, but difficult to master, forming an ideal basis for game interaction. There are only a limited number of basic emergent systems that are repeatedly used from game to game, and any new such system would be extremely welcome.

One of the most important emergent systems in current games is rigid body physics. It provides exactly the kind of intuitive but chaotic emergent system on which interesting gameplay is often based. It is now built in to most game engines and has fueled huge successes such as *Angry Birds*.

We think that another aspect of physics is underrepresented in current games: water simulation. People have an intuitive feel of how water masses flow over terrain and how objects float along currents. On the other hand, water exhibits some extremely complex behaviour. Water is a common element in childrens' play, e.g., building dams or floating ships through brooks. Therefore, water simulation could provide an interesting emergent system for gameplay.

There are some 2D games like *Sprinkle* and *Where's My Water* that utilise water as a gameplay element, but in most 3D games, water is just a static decoration without much interaction. For example, water simulation was mentioned in the *Just Cause 2* advertisement, but the area covered by water was fixed and the simulation only creates waves on the surface for a visual effect instead

Author's address: T. Kellomäki; email: timo.kellomaki@iki.fi.

https://doi.org/10.1145/2700533

of actually affecting gameplay. The simulation is also turned off on low-end machines, apparently being too computationally intensive. The only major 3D games we are aware of where water simulation forms a significant part of gameplay are *From Dust* and *Cities: Skylines*.

Water simulation itself is nothing new. Accurate models to simulate fluids have existed in engineering for decades. There is a vast literature of various techniques to solve the *Navier-Stokes equations* (NSE) that govern fluid dynamics. However, the equations are notoriously hard to solve even approximately, and as a consequence, a full 3D fluid dynamics solution is still beyond the frontier of possibility in current games. Therefore, extreme simplifications have to be made. Another problem posed is that mastering the complex simulation methods requires a lot of resources from the developers wishing to use water simulation in their games.

One of the most important simplifications is using heightfields. In these methods the water surface is represented simply by height values on a two-dimensional grid. In essence, one of the three dimensions is removed, which typically drops the number of cells needed from $O(n^3)$ to $O(n^2)$ for a given simulation resolution. However, the possibility to represent many interesting water phenomena are lost: splashes, breaking waves, and waterfalls all require more than a single heightfield.

The limitations of heightfield approaches are most obvious when water interacts with moving rigid bodies, since even a simple submerged body breaks the heightfield assumption. Current methods mostly provide solutions in limited cases, such as boats on a calm ocean.

2 REQUIREMENTS IN GAMES

Efficiency is undoubtedly one of the main factors in a water simulation system, since games most often have a very limited resource budget and water simulation is notorious for requiring considerable amounts of processing power. Most research examples naturally use all resources that are available in a system, but a game simulation would only be allowed to use a fraction of those. This means we need to bear in mind that instead of just real-time, the required speed should be at least around 3 to 20 times faster, depending on how essential a part of the game the simulation is.

Many solvers can be run in a highly parallel fashion, which makes using graphics processing units (GPUs) ideal for these purposes. This provides extra challenges for the methods that require complex data structures, since implementing them on the GPU is far from trivial.

Ease of use is largely a question of tool availability. Rigid body physics has been technically feasible in games for ages, but it only became widespread in all kinds of games after easy to use, off-the-shelf physics engines became widely available. The tools need to be robust and easy to set up, configure and use and be as flexible as possible. In general, the simpler the method itself is, the easier should making good tools be.

Quality of Experience (**QoE**) is an important factor. While almost all of the research is focused on achieving a realistic result, plausibility is typically more important than pure realism in games. As QoE is a subjective matter, comparing the methods in this regard requires user studies. However, the only such study we are aware of is a limited one by Kellomäki and Saari (2014b).

Instead of user studies, missing types of behaviour can function as a crude approximation of QoE. As an example, most users would not notice if water flows somewhat too quickly, but losing interesting phenomena like waterfalls and splashes because of heightfield limitations probably would harm the experience. An example of a non-visual but also extremely limiting case is forcing the boundaries of the water to be static during play, since it takes away many if not most of the interactive possibilities.

The visual part of QoE is somewhat independent of the solver used, since procedural techniques can be used to add visual details. They only need to rougly correspond to the underlying

simulation. With these methods and some shader effects even the simplest simulation might look good enough for games, because procedural details can be used to mask out the fact that the simulation is missing much of the realistic water behaviour. For examples of such techniques, see Chentanez and Müller (2010), Chentanez and Müller (2011), Yu et al. (2009).

Robustness is important for games. The NSE are continuous, but the solvers discretise time to finite steps. Increasing the time step will make the simulation run faster, but also causes artifacts and ultimately instability, which could ruin the whole experience if game mechanics depend on the simulation. Therefore a time step that is maximal, yet always stable, is usually selected.

In numerical engineering, stability is defined in terms of error compared to the correct solution of the equation being solved. However, as mentioned above, in games it is not usually important to find the correct solution. Therefore a more practical (if imprecise) definition of stability is in order. We use the term to refer to situations where the result looks obviously unnatural.

Stability is also directly linked to performance, since sometimes more complicated simulation schemes are more robust and thus the time step can be increased, compensating for the slower method. Therefore the methods should not be compared simply by how long a simulation step takes (or the inverse of it, the framerate), as is sometimes done. The ratio of simulation speed to realtime, which takes the time step into account, is more informative and will be used in this text.

Coupling with rigid bodies is essential for most imaginable games that use water as a game mechanic, since it forms the core of the interaction with water. The coupling should ideally be two-way: water needs to apply forces to objects so that they float, and are slowed by drag. On the other hand, water also has to bypass the objects (not flow into them) and react with splashes and waves when an object is in contact with it.

Rigid body coupling is also linked with robustness, since none of the current real-time methods can handle the coupling in all possible situations, and the coupling is often a source of instability, especially when fast-moving objects are involved.

3 CURRENT SOLVERS

The NSE have been known for centuries, but they are not analytically solvable in any practical situation. Most methods for solving the NSE numerically are aimed for non-realtime purposes: either engineering applications such as bridge design, or visual effects for movies. They are typically extremely slow. Simulating a single timestep can take from seconds to hours. Luckily, the degree of realism is unnecessary for most game purposes. We concentrate on such simplified methods that could be used in games. For a numerical solution our domain needs to be discrete. There are two main approaches to discretisation: grids (Eulerian approach) and particles (Lagrangian approach).

3.1 Grid-based Simulation

Grid-based simulation usually calculates the necessary values in thousands or millions of cells in a regular grid. If the number of cells per dimension, n, is increased, either the simulation resolution is increased or the simulated area becomes larger. If small-scale details can be created procedurally, there is no need to have a grid that is denser than around tens of centimeters to some meters per cell. Due to this limit, the faster techniques mainly enable simulating a larger area. For lack of a better term, we will use "resolution" to mean the number of cells per dimension (e.g., $n = 1024$).

The standard Eulerian simulation consists of advecting physical quantities along the flow, projecting pressure, and tracking the surface. Pressure projection is often the slowest part, but is required to achieve the realistic vortices caused by incompressibility (Bridson and Müller-Fischer 2007). There is a vast literature on making the method faster and more accurate, but the full 3D

case is still far from usable in real-time games, since the number of cells scales $O(n^3)$ for a given resolution. Moreover, the pressure solver is slower than $O(m)$ where m is the total number of cells.

There exists an interesting 2D/3D hybrid method utilizing so-called tall cells. A recent implementation is provided by Chentanez and Müller (2011). They use an adaptive grid where the vertical axis has several cells near the surface and a single tall cell for the bottom. This method preserves most of the interesting 3D phenomena while still reducing the number of cells to $O(n^2)$. The quality is undoubtedly the best of those reviewed in this paper, but the method is also somewhat expensive because of the semi-3D grid. As an example, their lighthouse scene uses a grid of $128 \times 128 \times 34$, where the last dimension is the vertical direction. The simulation using an NVIDIA GTX 480 card and a time step of 33 ms takes about 14 ms giving 2.4 times real-time performance (Chentanez and Müller 2011). The described method is complex, because the grid is adapted on the fly. However, the time complexity of their solver is linear to the number of cells, making the method asymptotically as fast as the fastest methods reviewed in this paper.

Faster methods can be achieved using a heightfield, which is essentially a 2D simulation. It therefore needs only $O(n^2)$ cells for a given resolution. Most heightfield algorithms have a time complexity of $O(m)$, where m is the total number of cells.

The shallow water equations (SWE) introduced to computer graphics by Layton and van de Panne (2002) are often used. They are reached from the NSE by assuming that velocity and pressure gradient are both constant along the vertical axis. For water, we also assume that the viscosity is zero. This model essentially only simulates a thin layer of water, but the result is good enough for many purposes. In this model the equations themselves are easier to solve and we can omit the time-consuming pressure projection and surface tracking steps. (Thürey et al. 2007).

A good example using the SWE is by Chentanez and Müller (2010). They solve the SWE on the GPU (NVIDIA GTX 480), achieving a timing of 18 ms for a grid size of 900×135 using a time step of 20 ms for a scene where a boat is ridden down a river. This gives a real-time factor of 1.1. Visual quality is assured by adding small-scale details using an FFT-based method. They also use a small number of particles to add several 3D effects like waterfalls and splashes, which is an interesting addition that any of the heightfield-based methods could employ to relatively cheaply restore some 3D behaviour.

An even faster method exists, called the pipe model. The method was pioneered by Kass and Miller (1990), who simplified the SWE further to reach the 2D wave equation on the water surface. O'Brien & Hodgins (1995) formulated the method intuitively using virtual pipes. The flow is modelled by pipes between columns of water on the grid. Hydrostatic pressure is used to calculate the flow in each pipe. The result lacks vortices, but is quite convincing, considering its simplicity, especially when run over a heightmap terrain. Our implementation of the pipe model is examined in more detail in Section 5.

Many game developers have toyed with the pipe method since the nineties and a variant of it was also used commercially in *Cities: Skylines* (2015). As the current programmable GPUs are getting easier to use for general-purpose computing, tools or libraries that make the method easy to use on the GPU could be realistically made, finally making widespread adoption of water simulation in games possible.

The Lattice Boltzmann method (LBM) is another grid-based simulation method suitable for water simulation. The 3D versions are realistic and handle rigid body coupling well, but just as other 3D methods, they are not fast enough for games (e.g., Thürey et al. 2006). On the other hand, 2D versions of LBM are very fast and have been used to simulate deep water waves with blocking geometry in real time (Geist et al. 2010) and for real-time shallow water simulation on a heightfield terrain (Ojeda 2013).

There are also various methods that concentrate on the surface waves, such as the FFT-based family of methods by Tessendorf (1999). For a survey on such methods, see Darles et al (2011). However, these methods don't allow the water to flow anywhere. While very realistic-looking visualizations of especially oceans can be rendered with them, from a gameplay point of view they are of limited use, and fall outside the scope of this article.

3.2 Particle-based Methods

Lagrangian or particle-based methods track the relevant quantities at moving locations. They are usually fully 3D and therefore require $O(n^3)$ particles to achieve a given resolution. The upside is that particles are automatically adaptive: They are only needed where water is present, so the simulated area can be of any size as long as the amount of water is limited. In grid-based methods, either complex adaptive grids are needed or the whole game world needs to be simulated. This means that the less water there is, the better the particle-based methods become in comparison to grid-based methods.

Particle simulation is already sometimes used in game effects which create only small amounts of water, such as a broken pipe creating a puddle on the floor. On the other hand, simulating whole lakes and rivers using SPH requires a huge number of particles, because most of them would be sitting uselessly deep under the surface. Also, the performance of particle-based methods is unpredictable, since the number of particles changes with the amount of water. For games, the worst-case performance is typically very important, because what happens in the game is unpredictable, yet the game has to run smoothly in every situation.

Probably the most common Lagrangian method is *smooth particle hydrodynamics* (SPH), which is again based on the NSE. The physical quantities, such as pressure, are tracked at varying particle locations instead of a fixed grid. To find the value of a quantity in a given location, a weighted sum of the quantity in the particles is calculated. The weight decreases with distance. This would make the algorithm $O(m^2)$ in the number of particles m (which would then be $O(n^6)$ for a given resolution). This can be counteracted by letting the weighting function go to zero relatively quickly and using space-partitioning data structures, resulting in an $O(m)$ time complexity. (Bridson and Müller-Fischer 2007).

The NSE are greatly simplified by the SPH approach. All that remains is basically determining the forces that affect each particle. The effect of viscosity and pressure is caused by the other particles and can be calculated by iterating through them. External forces can be simply applied to each particle independently of the others. (Bridson and Müller-Fischer 2007).

Because SPH does not provide a directly renderable surface, drawing could take more time than the simulation itself. Since the number of particles is limited, they also need to be smoothed somehow, lest the individual particles be seen. One solution is to find an isosurface of the density field and triangulate it with, e.g., the marching cubes algorithm (Bridson and Müller-Fischer 2007). Kipfer and Westermann (2006) developed a method using a 2D grid where the height is displaced by the density at each grid point to simulate large amounts of water, but this discards many of the advantages of having a full 3D solution without all the performance gains of also simulating only a heighfield. An interesting solution by van der Laan et al. (2009) is to use screen-space rendering.

Since SPH is a full 3D method, it is relative easy to couple with rigid body simulation. This topic is further discussed in Section 4.

SPH is also easy to integrate in the current physics engines, which already support particles. Of the currently popular physics engines, at least *NVIDIA PhysX* implements SPH. This tool support has enabled its use as a gameplay element in some 2D games and for small-scale visual effects in some 3D games, but as a particle method it is not well suited for games where large areas could be dynamically covered by water.

The research community in Lagrangian methods is very active and has produced considerable improvements to the methods in recent years. For example, the position-based fluids introduced by Macklin and Müller (2013) allow for much larger time steps than the previous methods. For a survey on SPH and related methods, see Ihmsen et al. (2014).

In an interesting development, SPH and SWE have been combined by Solenthaler et al. (2011). This drastically reduces the number of particles needed, as only the surface of the 2D heightmap is simulated using SPH. The resulting particle density is then interpreted as water height. However, their GPU implementation takes about 25 ms using 128k particles and a time step of 2 ms on an NVIDIA GTX 460. This only gives a real-time factor of 0.08. In addition, this only includes the simulation part, disregarding the time-consuming drawing step.

On the other hand, judging by the timings, it seems that the implementation of Solenthaler et al. (2011) does not use any space-partitioning data structures, resulting in an $O(m^2)$ time complexity. Implementing good data structures for this problem efficiently on the GPU is relatively complex but has been done for 3D SPH by several researchers (e.g., Harada et al. 2007; Goswami et al. 2010). Since most SPH researchers do not disclose their time step, performance comparison is not possible. It would also be interesting to see the recent developments in the 3D Lagrangian methods to be applied to this idea.

4 COUPLING WITH RIGID BODIES

As mentioned earlier, two-way coupling with rigid bodies is an essential feature for any water simulation method aiming to be widely usable in games.

Coupling is a popular topic among researchers working with full 3D methods. In 3D Eulerian methods that solve the NSE, rigid bodies are traditionally handled as boundary conditions for the equations (Benson 1992). Water-to-body coupling is achieved by integrating pressure along the object borders. Research focuses on more difficult cases such as thin and deformable objects (e.g., Guendelman et al. 2005) or more physically-based modeling (e.g., Robinson-Mosher et al. 2009).

SPH is also a full 3D solution. Coupling methods in this literature are often based on sampling the bodies as particles and assigning penalty forces to the particles based on distances, which works in both coupling directions. Also in this field, most current research concentrates on the difficult cases. (Akinci et al. 2012).

Heightfield methods, however, run into trouble with the coupling. The essential assumption of the heightfield is that water is continuous in the vertical direction. A simple submerged body already breaks this assumption, not to mention several moving bodies that can have air pockets underneath. Clearly, a general coupling solution is not possible in the heightfield framework. Either the heighfield assumption needs to be relaxed or we need to settle for a less general solution. This may be the reason for the apparent lack of research on this topic: There are only a few results using relatively simple approaches, at least compared to how popular the topic is in the full 3D field. The current heightfield solutions are also far from being physically-based.

The most common solution is to confine to a limited subset of situations. As an extreme example of this approach, consider one-way coupling, where the water is unaffected by the bodies. Now, for a floating body, the water surface passes right through the object. This enables calculating the buoyant force simply based on the submerged volume. Similarly, the drag is easy to calculate by comparing the water and object velocities for each face of the object. The calculations can be based directly on the object triangles (e.g., Chentanez and Müller 2010) or on planted probes on the floating objects (e.g., Cords and Staadt 2009). If water is allowed to flow through the bodies, this direction is usually handled well even by real-time heightfield methods.

The body to water effect can be roughly divided into two distinct parts. Firstly, objects cause waves on the surface. These waves are then propagated to the surroundings. This is often enough for visualizing situations like boats on open waters. Secondly, the bodies block flow, e.g., when dams are built out of dynamic bodies or a wave hits large objects.

The methods employed to create believable waves emanating from a floating object are usually only loosely physically-based. Water usually still exists inside the floating bodies. In many solutions, waves also pass through the objects totally or almost unaffected, which looks unnatural in the case of large objects.

O'Brien and Hodgins (1995) created a model for the situation where an object hits the water surface. They estimate a force so that the entering mass is situated on the surface. This causes an external pressure, which makes the water flow to the neighboring cells. However, their method does not take the volume of the body into account. They also introduced particles to add splashes above the heightfield. Mould and Yang (1997) use a similar model. The method causes believable waves to be emanated from the collision point, but existing waves still pass through the objects.

Thürey et al. (2007) take the volume of bodies into account. They first mention pushing the columns down until overlaps are removed and redistributing the removed water in the neighbourhood. They note that this method could not handle submerged bodies and continue to develop a method where the underwater volume of bodies is tracked in each cell. Changes in this volume cause water to be moved between neighbours. We have implemented this method and evaluate it in Section 6.

Yuksel et al. (2007) take a completely different approach. They represent the waves that are created as *wave particles* that are propagated independently of each other. Collisions between existing waves and objects are not handled. Instead, new waves are created so that the existing ones are countered by the superposition principle. This way waves do not appear to go through objects. This method works remarkably well in open waters, where the flow aspect is not very important.

Another option might be to generalise the heightfield. Mould and Yang (1997) already generalised the virtual pipe method to have several layers of water on top of each other. Kellomäki (2014a) uses this idea by splitting the heighfield to two layers, one under and one above an object in the water. The method concentrates on blocking the flow by modifying the virtual pipe method. Water never enters the bodies, unlike in the previously considered solutions. This makes it possible to simulate many new kinds of situations. For example, dams can be built out of the rigid bodies and flow can be diverted around and under large, dynamic objects. The method assumes that the blocking bodies are contiguous in the vertical direction to make two layers sufficient. This method is also evaluated in Section 6.

The method proposed by Kellomäki (2014a) has its own problems. The assumptions are rather severe, which limits the usefulness of the method. The method also suffers from spatial aliasing problems because each cell is either completely blocked or completely free. Lastly, the method is more complicated and less performant than the non-blocking heighfield methods.

Particles have already been used in conjunction with grid-based simulation for other phenomena that are not representable by heightfields. It could be possible to seamlessly convert water into particles and back near blocking bodies and use SPH (or its derivatives) to simulate those areas. Well-known existing coupling methods for SPH could then be used. SPH is a standard method that has proven to be an excellent solution for simulating small amounts of water. Therefore, we find this kind of hybrid method to be a promising approach. However, making the transition region between grids and particles invisible to the user is potentially a demanding task. The resulting hybrid method would also be much more complicated than the current ones.

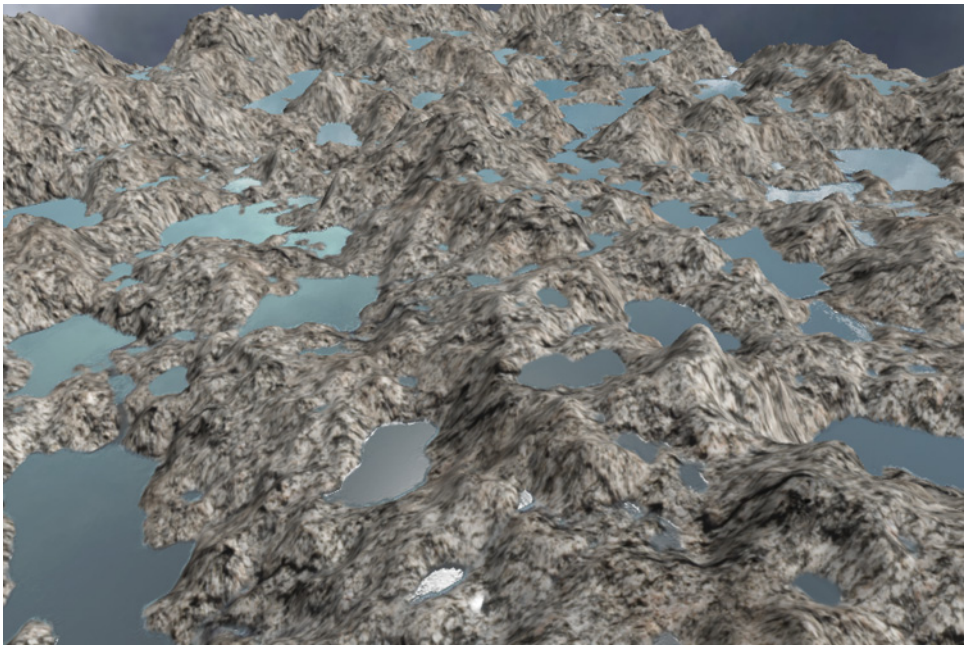

Fig. 1. Water simulation using the pipe method creates lakes naturally.

5 OUR PIPE MODEL IMPLEMENTATION

We have implemented a lightweight water simulation using the pipe model (Figures 1, 2, 3). Our solution is directly based on that of Mei et al. (2007), thus no details of the method itself are presented here.

In our simulation, water spreads naturally to new areas, which can be used as a game mechanic. The implementation handles massive amounts of water on a very large grid. To demonstrate interaction, the user is able to raise and lower the terrain on the fly. This can be used for example to block water from reaching certain areas, which is an example of gameplay built on top of the simulation.

We implemented the simulation completely on the GPU using OpenGL fragment shaders that read from and write to textures. The first shader adds water at user-specified sources or everywhere in case of rain. Another shader calculates the flows between neighbouring cells based on previous flows and water levels. Finally, a third shader updates the heights using these new flows. The phases are illustrated in Figure 4. The read/write operations are implemented as usual by alternating between two textures.

The heightmap is simply drawn as a triangle strip. The heights are interpolated between two last simulation steps to allow high frame rates with relatively slow updates. We use a basic water shader with reflection from the skybox and a Fresnel term for translucency. In addition, the simulation method produces flow velocity at each grid point, which is used to distort two alternating normal maps similarly to the approach described by Vlachos (2010). This adds small details that seem to flow with the current, greatly increasing the visual plausibility.

The pipe model does not advect velocities which means the result lacks vortices. However, there was no preference difference found in a user test where such an approach was compared to full SWE simulation with an advection step (Kellomäki and Saari 2014b). Many visual additions, such as particles for splashes and waterfalls, foam, etc. employed by the implementations mentioned in this paper could be added to increase plausibility and visual quality.

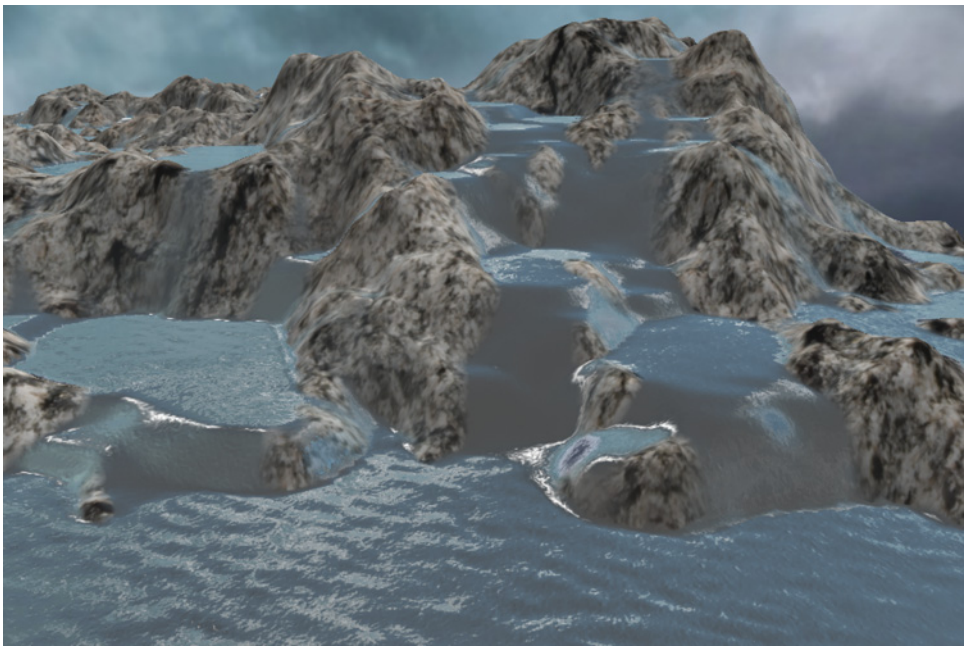

Fig. 2. Water flowing down the hills.

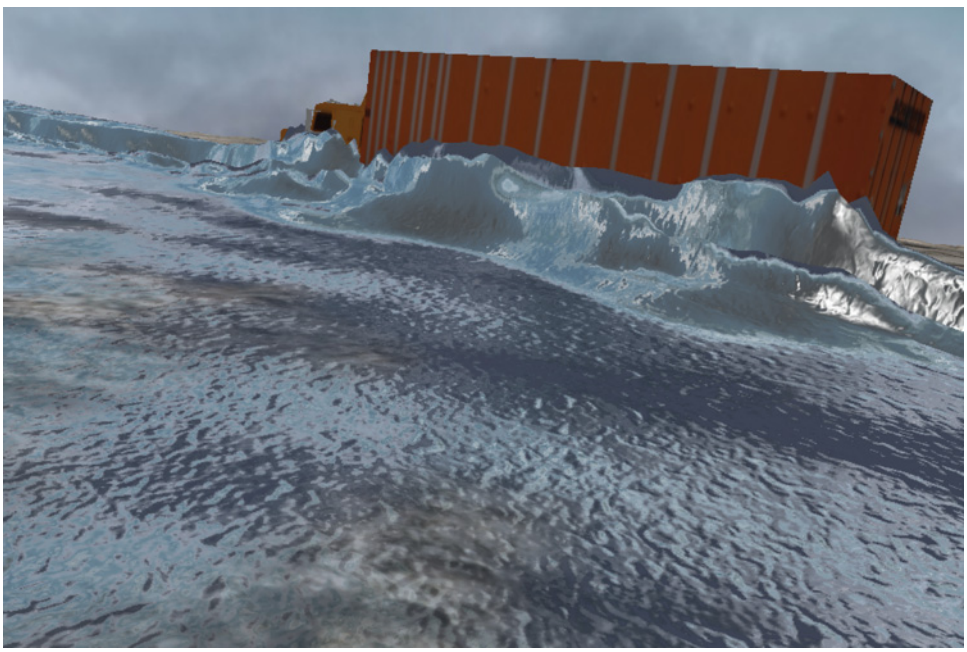

Fig. 3. A large wave hitting a truck.

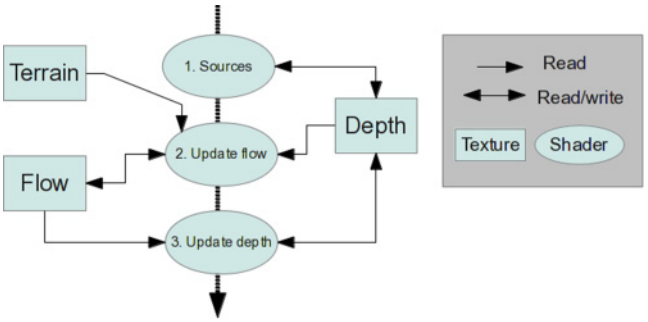

Fig. 4. A flow diagram of our pipe method implementation.

Table 1. Timings of the Pipe Method Example. rtkc (Real-Time Kilocells) is the Number of Cells in Thousands That Can be Simulated Real-Time.

Grid size	Sim. time (ms)	× realtime	rtkc
2048 × 2048	7.1	3.5	14700
1024 × 1024	3.3	7.6	8000
512 × 512	1.1	23	6000
256 × 256	0.55	45	3000
(empty)	0.41	61	N/A

Timings were measured on a laptop using an NVIDIA Quadro 1000M card, which is significantly slower than those used in the other examples mentioned in this paper. We use a timestep of 25 ms, which we experimentally found to be stable even in extreme situations. Selecting the timestep poses a trade-off between speed and instability. Each researcher has most likely selected the maximal timestep that makes their method stable enough. Thus, stability differences are mostly already included in the performance figures.

Like most of the other methods, the virtual pipe method has $O(m)$ time, memory bandwidth, and memory requirements, where m is the number of cells and needs $O(n^2)$ cells for a given resolution. In practice, the performance of our implementation is bound by texture bandwidth of the GPU. The shaders consist of very simple arithmetic operations, but must sample several quantities of their neighbours for each cell in several consecutive phases.

In this paper, we use the number of cells or particles (in thousands) that can be simulated real-time (real-time kilocells, *rtkc*), to measure the performance of a simulation method. The number of cells or particles is in practise proportional to the area that can be simulated. Results for our pipe model implementation are summarised in Table 1.

The *(empty)* row in Table 1 is for estimating the overhead that is independent of the grid size. Larger grids seem to be faster because of the constant overhead, but if it was removed, rtkc would demonstrate roughly the theoretical linear time complexity of the pipe model.

A comparison of the performance of different methods is presented in Table 2. We use our smallest grid size (256 × 256) for comparison, since the references do not give timings for larger grid sizes (due to them not being realtime with the slower methods). The results are not directly comparable, because different hardware was used, the timings are not done with the same methods, and scene contents and grid sizes are varying. Also, the comparison between Eulerian and Lagrangian methods is not completely fair. The number of particles in SPH is not directly comparable to the

Table 2. Performance Comparison of Different Methods. Cells is the Number of Horizontal Cells or Particles as Appropriate. RTKC Means Real-Time Kilocells.

Method	Cells	$\triangle t$	$\times$ timing	rtkc
Tall cells (Chentanez and Müller 2011)	16k	33 ms	33 ms	16
SWE (Chentanez and Müller 2010)	122k	20 ms	2.2 ms	1100
2D SPH (Solenthaler et al. 2011)	128k	2 ms	50 ms	5
2D LBM (Ojeda 2013)	16k	16 ms	0.35 ms	730
Pipe method	65k	25 ms	0.55 ms	3000

number of grid cells. Since particles are naturally adaptive, a given number of particles gives better results than the same number of cells if there are enough dry areas where no particles are needed. For the tall cell method, the vertical dimension is simply dropped, since it does not increase the area that can be simulated. In other words, the 34 vertical cells are only used to add quality via realistic 3D surface phenomena. On the other hand, both the hardware and the selected grid sizes do a disfavour to the pipe model. Still, we feel that this table gives an interesting rough idea of the relative performance of the methods, since the differences are quite large.

All three grid-based methods have the same asymptotic time complexity, but the pipe model is still significantly faster than the other methods. With grid sizes of 512×512 and upwards the only methods currently suitable for actually simulating large amounts of water in games are the pipe method, SWE and 2D LBM. It would be especially interesting to compare the pipe method results with LBM in the future.

The particle-based methods can compete if the number of particles stays low, but that limits the amount of water. The pipe model is also simpler to implement than any of the alternatives, which contributes to the ease of use and will make it easier to create quality tools for game developers.

6 COMPARING RIGID BODY COUPLING METHODS

To make another comparison, we have also implemented two versions of rigid body coupling in our simulator: one with focus on water-to-object coupling (Thürey et al. 2007) and another with focus on object-to-water coupling (Kellomäki 2014a). The two coupling implementations are compared in Figure 5, where a cube is blocking an approaching wave. For the scenario of dropping an object into calm water, the methods use a similar idea to each other and the result is almost identical.

With the method of Thürey et al. (2007) the objects float and are moved by the current very convincingly. Flow velocities provided by the pipe method are used for the advection and floating is implemented by simply calculating the submerged volume in each simulation cell. The strength of the object-to-water effect can be adjusted using a parameter, but setting a large value causes instability with our long time steps. With stable parameters, either the waves caused by the object are rather small or the performance is considerably worse. The method does not prevent water from entering the bodies and therefore the bodies do not block flow. This method seems good for small and light objects floating in situations without too much flow. In our implementation, almost all of the coupling work is done on the CPU that would otherwise be almost idle, so there is no real slowdown in this framework.

Using the method proposed by Kellomäki (2014a) the flow is blocked naturally. There are some visual problems due to cell aliasing for floating objects. Our implementation of this method is GPU-based and runs at roughly one quarter of the speed of the original pipe method without coupling. The performance is almost independent on the number of bodies, since the same shader code is evaluated whether or not any objects are present in a given cell. It would be possible to use another

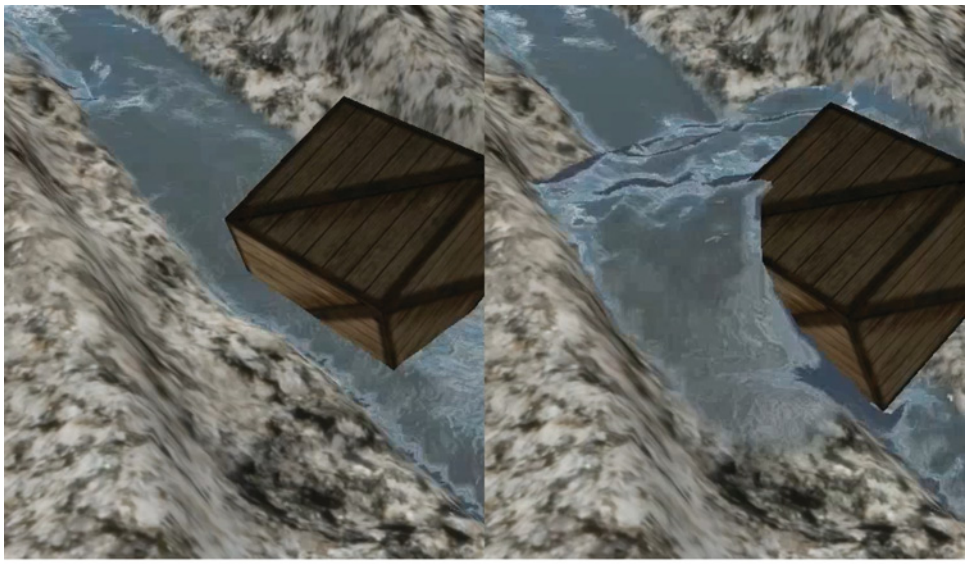

Fig. 5. The effect of object-to-water coupling: a box blocking flow.

approach where the more complex coupling shader is only evaluated in areas with bodies present, which would probably improve average performance, but increase the number of draw calls and make the performance dependent on the number of objects. An example of the waves caused by dropping a cube into water is given in Figure 6.

One of the bottlenecks in both of our coupling implementations are that we use the CPU implementation of the Bullet physics engine to handle the rigid bodies. The coupling requires reading back water levels to the CPU. The GPU implementation of Bullet could in principle be used to offload the whole simulation to the GPU to minimize amount of readbacks. In practise the readbacks cannot be completely avoided, because the application logic will most probably need the information, but using information that is delayed a couple of frames solves most of the performance problems related to that.

Neither of the coupling methods we implemented or evaluated is even close to good enough for a generic water engine for games. Such an engine should be versatile enough for many kinds of situations. For now, we have to be satisfied with limited scenarios. For example, open water scenes with floating objects can already be implemented believably by the Thürey et al. (2007) method or wave particles Yuksel et al. (2007). For practical examples of wave particles in action, see *Uncharted 3* (Gonzalez et al. 2012) and *Assassin's Creed 3* (Seymour 2012).

7 CONCLUSION

Water simulation could provide games with interesting, fresh sources of emergent interaction. Thus far it has only been used in a very limited number of 3D games. We suspect that the most important hurdles to cross are performance, ease of use, and the lack of a robust method for rigid body coupling for heightfield-based methods.

We compared some of the more complex methods and our own implementation of the simple pipe model. The pipe model is the fastest of the methods and especially well suited for games with large water areas. It also produces acceptable QoE despite the lack of realism in the simulation.

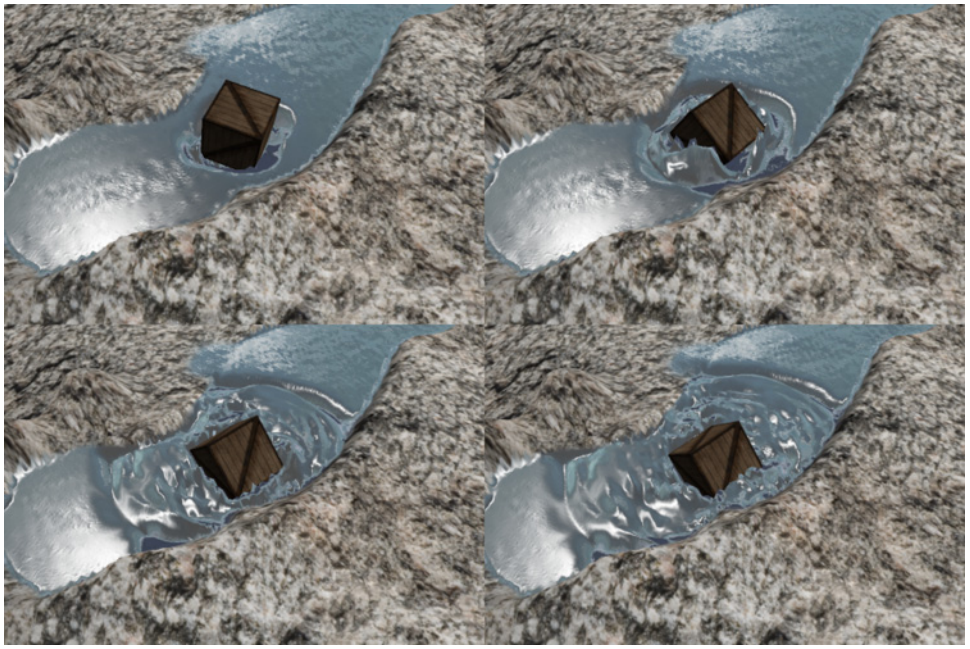

Fig. 6. Another example of object-to-water coupling: a box is dropped in a calm lake.

Most deficiencies of the method could be masked using modern shader effects and other tricks, such as using a small number of particles to create splashes. The pipe method is also the simplest one of the surveyed methods, which makes it the easiest to implement and use.

Rigid body coupling remains one of the most difficult problems in heightfield methods. For some game types (especially if water spreads on an uneven terrain), this is a serious problem, while other situations, such as open water scenes, can already be handled well enough. In the future, we expect to see grid-particle hybrid methods that take the interaction to a new level.

To make water simulation easy enough for widespread adoption, further tool development is needed. Just as is the case with rigid body physics, the application programmer ought to be able to create a game utilising water simulation without needing to understand the physics profoundly.

As further research, it would be interesting to implement the various simulation methods in the same framework, possibly even in an actual game example. This would enable more realistic and fair performance comparisons, and, on the other hand, make it possible to conduct large-scale user testing to assess the QoE for different methods.

REFERENCES

N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, and M. Teschner. 2012. Versatile rigid-fluid coupling for incompressible SPH. In *ACM Transactions on Graphics (TOG)* (31, 4, 62). ACM.

D. Benson. 1992. Computational methods in Lagrangian and Eulerian hydrocodes. In *Computer Methods in Applied Mechanics and Engineering* (99, 235–394). 1992. Elsevier.

R. Bridson and M. Müller-Fischer. 2007. Fluid simulation: SIGGRAPH 2007 course notes. In *2007 Courses, SIGGRAPH'07* (1–81). ACM.

N. Chentanez and M. Müller. 2010. Real-time simulation of large bodies of water with small scale details. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (197–206). Eurographics Association.

N. Chentanez and M. Müller. 2011. Real-time eulerian water simulation using a restricted tall cell grid. In *ACM Transactions on Graphics (TOG)* (30, 4, 82). ACM.

H. Cords and O. Staadt. 2009. Real-time open water environments with interacting objects. In *Proceedings of Eurographics Workshop on Natural Phenomena (EGWNP) (2)*. Eurographics Association.
E. Darles, B. Crespin, D. Ghazanfarpour, and J.-C. Gonzato. 2011. A survey of ocean simulation and rendering techniques in computer graphics. *Computer Graphics Forum*, 30, 1, 2011.
R. Geist, C. Corsi, J. Tessendorf, and J. Westall. 2010. Lattice-boltzmann water waves. In *Advances in Visual Computing, Lecture Notes in Computer Science* (6453, 74–85). Springer.
C. Gonzalez Ochoa and D. Holder. 2012. Water technology of uncharted. *Presentation in Game Developers' Conference* 2012.
P. Goswami, P. Schlegel, B. Solenthaler, and R. Pajarola. 2010. Interactive SPH simulation and rendering on the GPU. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (55–64). Eurographics Association.
E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. 2005. Coupling water and smoke to thin deformable and rigid shells. In *ACM Transactions on Graphics (TOG)* (24, 3, 973–981). ACM.
T. Harada, S. Koshizuka, and Y. Kawaguchi. 2007. Sliced data structure for particle-based simulations on GPUs. In *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia* (55–62). ACM.
M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner. 2014. SPH fluids in computer graphics. In *Eurographics 2014 – State of the Art Reports*. The Eurographics Association, 2014.
M. Kass and G. Miller. 1990. Rapid, stable fluid dynamics for computer graphics. In *ACM SIGGRAPH Computer Graphics* (24, 4, 49–57). ACM.
T. Kellomäki. 2014a. Two-way rigid body coupling in large-scale real-time water simulation. In *International Journal of Computer Games Technology*, 2014.
T. Kellomäki and T. Saari. 2014b. A user study: Is the advection step in shallow water equations really necessary? In *Eurographics Short Papers*, 2014.
P. Kipfer and R. Westermann. 2006. Realistic and interactive simulation of rivers. In *Proceedings of Graphics Interface 2006* (41–48). Canadian Information Processing Society.
A. T. Layton and M. van de Panne. 2002. A numerically efficient and stable algorithm for animating water waves. In *The Visual Computer* (18, 1, 41–53). Springer.
M. Macklin and M. Müller. 2013. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32, 4, 2013.
X. Mei, P. Decaudin, and B. G. Hu. 2007. Fast hydraulic erosion simulation and visualization on GPU. In *15th Pacific Conference on Computer Graphics and Applications* (47–56). IEEE.
D. Mould and Y. H. Yang. 1997. Modeling water for computer graphics. In *Computers & Graphics* (21, 6, 801–814). Elsevier.
J. F. O'brien and J. K. Hodgins. 1995. Dynamic simulation of splashing fluids. In *Proceedings of Computer Animation'*95 (198–205). IEEE.
J. Ojeda. 2013. *Efficient Algorithms for the Realistic Simulation of Fluids*. PhD thesis, FIB, 2013.
A. Robinson-Mosher, R. E. English, and R. Fedkiw. 2009. Accurate tangential velocities for solid fluid coupling. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (227–236). ACM.
M. Seymour. 2012. Assassin's creed III: The tech behind (or beneath) the action. *Article in fxguide*. URL: http://www.fxguide.com/featured/assassins-creed-iii-the-tech-behind-or-beneath-the-action/ (cited February 13, 2013).
B. Solenthaler, P. Bucher, N. Chentanez, M. Müller, and M. Gross. 2011. SPH based shallow water simulation. In *Workshop in Virtual Reality Interactions and Physical Simulation* (39–46). The Eurographics Association.
J. Tessendorf. 1999. Simulating ocean water. *SIGGRAPH Course Notes*, 1999.
N. Thürey, U. Rüde, and M. Stamminger. 2006. Animation of open water phenomena with coupled shallow water and free surface simulations. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (157–164). Eurographics Association.
N. Thürey, M. Müller-Fischer, S. Schirm, and M. Gross. 2007. Real-time breaking waves for shallow water simulations. In *Computer Graphics and Applications, Proc. of the 15th Pacific Conference on Graphics* (39–46). IEEE.
W. J. van der Laan, S. Green, and M. Sainz. 2009. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (91–98). ACM.
A. Vlachos. 2010. Water flow in Portal 2. In *ACM SIGGRAPH 2010 Courses* (1–54). ACM.
Q. Yu, F. Neyret, E. Bruneton, and N. Holzschuch. 2009. Scalable real-time animation of rivers. In *Computer Graphics Forum* (28, 2, 239–248). Blackwell Publishing Ltd.
C. Yuksel, D. H. House, and J. Keyser. 2007. Wave particles. *ACM Transactions on Graphics (TOG)* (26, 3, 99). ACM.

Received February 2013; revised October 2014; accepted February 2016