Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2018)
F. Jaillet, G. Zachmann, K. Erleben, and S. Andrews (Editors)

Real-Time Virtual Pipes Simulation and Modeling for Small-Scale Shallow Water

F. Dagenais[1], J. Guzmán[1], V. Vervondel[1], A. Hay[2], S. Delorme[2], D. Mould[3] & E. Paquette[1]

[1]École de technologie supérieure, Canada [2]OSSimTech, Canada [3]Carleton University, Canada

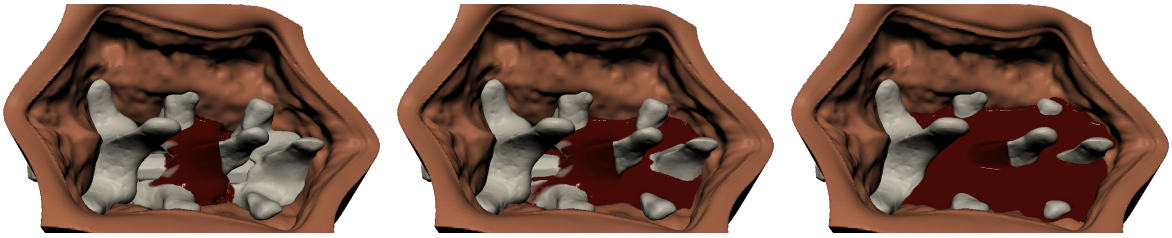


Figure 1: *Frames from an animation where blood flows from a vertebra. Note the multiple overhangs and holes.*

Abstract

We propose an approach for real-time shallow water simulation, building upon the virtual pipes model with multi-layered heightmaps. Our approach introduces the use of extended pipes which resolve flow through fully-flooded passages, which is not possible using current multi-layered techniques. We extend the virtual pipe method with a physically-based viscosity model that is both fast and stable. Our viscosity model is integrated implicitly without the expense of solving a large linear system. The liquid is rendered as a triangular mesh surface built from a heightmap. We propose a novel surface optimization approach that prevents interpenetrations of the liquid surface with the underlying terrain geometry. To improve the realism of small-scale scenarios, we present a meniscus shading approach that adjusts the liquid surface normals based on a distance field. Our approach runs in real time on various scenarios of roughly 10 x 10 cm at a resolution of 0.5 mm, with up to five layers.

CCS Concepts
•***Computing methodologies*** → ***Physical simulation;***

1. Introduction

In this paper, we focus on real-time simulation of shallow water at small scales, such as in scenarios of spilled coffee or bleeding during surgery. In such situations, thin layers of liquid flow on a surface and may also eventually fill up small cavities. At this scale, effects such as the viscous drag force exerted on the liquid by the surface of the obstacles, as well as the meniscus at the wet–dry boundary, are much more prominent.

In many real-time contexts, such as medical applications and games, it is necessary to have a very efficient simulation since other systems are running on the same resources (e.g., haptic feedback, other physics, AI). A full 3D simulation is often too expensive; only very coarse resolutions can achieve real time. However, a coarse 3D resolution fails to represent thin films of liquids, such as blood or paint flowing over a surface. To reduce computation times, some methods focus on performing a 2D simulation on a heightmap, such as Chentanez and Müller [CM10] and Mei et al. [MDH07]. Such methods can simulate liquids that are arbitrarily deep or shallow, with no impact on the resolution of the simulation.

Our approach builds upon the virtual pipes (VP) method [vBBK08]. Existing work does not use a physical model to handle the viscous drag force from the terrain. This force is non-negligible for various liquids such as blood and paint. To allow more complex terrain geometries that contain overhangs and holes, previous researchers extended virtual pipe methods to use a multi-layered heightmap [Kel14] and created interconnections between the layers to allow the liquid to flow between them. Such configurations are important for different scenarios, such as when blood should flow below organs in a surgery simulation. Nevertheless, current multi-layered methods do not handle the flow through

fully flooded passages below obstacles. When such passages get completely filled, the flow to these cells stops, preventing future flow while the passages remain filled. Additionally, current multi-layered VP methods have a limited surface representation, with some leading to discontinuities in the surface mesh, while others are unable to accommodate multiple overlapping layers. Finally, most related works typically aim for large-scale simulations and ignore the surface meniscus shading.

Our approach extends current multi-layered heightmap VP methods to enhance both the behavior and the shading. The behavior is improved using a physically-based viscosity model and by considering the flow below obstacles. Furthermore, our improved multi-layered surface reconstruction does not suffer from discontinuity issues. Our novel surface optimization approach provides a correction to the mesh surface, preventing interpenetrations with the underlying terrain geometry. We work on the simulation of moderate amounts of liquid, roughly in the 10 ml to slightly over 1 liter range. At this scale, our new meniscus shading approach significantly improves the visual results of our simulations. To summarize, our contributions are as follows:

- We present a physically-based stable viscous model that takes the fluid height into account.
- We propose the extended pipes interconnection framework allowing flow below obstacles.
- We perform independent multi-layered surface reconstruction with smooth boundaries.
- We use a surface optimization to prevent unwanted interpenetrations between the heightmap and an arbitrary mesh.
- We propose a distance-based correction of the normals to enhance the specular shading of the meniscus.

2. Related work

In the computer graphics field, most efforts for animating liquids are spent on offline simulations using either an Eulerian [EMF02], particle-based [IOS*14], or hybrid [JSS*15] simulation. Some work focuses computations on areas with more details using adaptive grid structures [AGL*17], narrow band surfaces [FAW*16], or adaptive particle radii [IOS*14]. These methods are capable of generating astonishing visual results, but are nevertheless too slow for real-time applications. Macklin and Müller [MM13] were able to achieve impressive visual results by simulating and rendering more than 100,000 particles in real time using their position-based dynamics framework. While this method is real-time, its ability to reproduce smooth thin layers of liquid is limited.

To reduce memory and computational costs, it is more efficient to use a heightmap to represent the fluid and perform a 2D simulation to update the liquid's height. While methods using the heightmap as such cannot exhibit some more complex behaviors such as splashes and wave crests, they are adequate for a broad range of scenarios. For example, intricate wave patterns in shallow water can be encoded as height displacements by particles [YHK07] or packets of similar wavelengths [JW17]. Furthermore, heightmap methods can simulate arbitrarily thin films of liquids with no impact on the simulation resolution. Lee and O'Sullivan [LO07] allowed some compressibility in their 2D simulation based on the Navier-Stokes equations and adjusted the liquid's height based on its density. While simple and efficient, this technique does not account for the underlying terrain elevation. By assuming a vertical anisotropy of the liquid's velocity, the Navier-Stokes equations can be simplified, resulting in the shallow water equations, which can be further simplified to the shallow wave equations [KM90]. Several papers focus on implicitly solving these equations on a regular grid [KM90, LvdP02] or on triangular mesh surfaces [WMT07, ATBG08]. Methods with an implicit integration maintain stability at larger timesteps, but are prone to a lot of diffusion as well as volume gain when using a large timestep. Furthermore, faster-moving boundaries require a smaller timestep, which can considerably increase the computation time. On the other hand, Chentanez and Müller [CM10] showed that their explicit integration of the shallow water equations is able to simulate large-scale scenarios in real time. For their part, our experiments show that the explicit integration requires a considerably smaller timestep for small-scale examples because of the larger ratio between the liquid's velocity and the simulation cell size, which limits its use for real-time applications in that context.

A simpler model for simulating shallow water, the VP method, was introduced by O'Brien and Hodgins [OH95]. It is based on the hydrostatic pressure difference between neighbor cells of a uniform grid. Liquid is transferred between them through virtual pipes connected at their bottom, and the simulation uses an explicit integration. This method has been extended to support multi-layered heightmaps in order to allow simulations above partially submerged floating obstacles [Kel14] and on more complex terrains with overhangs [BMPG11]. Furthermore, our experiments show that the VP method allows a considerably larger timestep than with the shallow water method of Chentanez and Müller [CM10] for small-scale scenarios. As such, our approach builds upon the VP method to simulate real-time, small-scale shallow waters. However, these methods have some limitations in in the case of small-scale simulations. First, they lack a physically-based viscosity model: they do not account for the viscous drag forces applied on the liquid. Furthermore, multi-layered methods completely block the liquid's flow under obstacles (e.g., overhangs), which results in an unrealistic behavior. For those reasons, we improve the behavior of the VP method by formulating a novel physically-based viscosity model and by considering the flow underneath fully-flooded passages.

Our goal is to simulate small amounts of liquids, with a millimeter to sub-millimeter resolution. Another feature is important at such scales: the meniscus, which is the effect of the capillary action at the fluid–solid boundary. Kerwin et al. [KSS09] propose a real-time meniscus shading method relying on a pixel-based edge detection. Although fast, this method delivers limited realism, as it does not take into account the size of the meniscus, nor does it differentiate between a concave and a convex meniscus.

Prior VP methods often ignore effects that are visible at a smaller scale, such as the viscous drag force from the terrain and the meniscus near boundaries. Furthermore, multi-layered frameworks neglect the flow underneath fully-flooded passages below obstacles. We thus extend the VP method with a physically-based model for the viscous drag force and we propose an extended pipe model that

handles the flow underneath obstacles. Furthermore, we improve the surface to account for its evolving multi-layer topology and we develop a new meniscus shading approach.

3. Liquid simulation

Our approach targets the real-time simulation of small amounts of liquid based on the VP method (Sec. 3.1). Our contribution breaks down into two categories: improving fluid behavior and improving the rendering. On the behavior side, we first propose a new formulation for the viscosity (Sec. 3.2), allowing a varying amount of viscosity and widening the range of behavior from water to liquids such as blood and paint. In a further improvement to the behavior, we add the ability to handle simulation domains containing overhangs and holes, extending the multi-layered VP method to account for the flow under obstacles through *extended pipes* (Sec. 3.3). Our rendering improvements include optimizing the surface to prevent various forms of artifacts (Sec. 4), and adjusting the surface normals to account for the meniscus (Sec. 5).

3.1. Virtual pipes simulation

This section explains how we combined different VP methods to derive our specific VP model. Our base liquid simulation follows the VP method formulation introduced by O'Brien and Hodgins [OH95], with the multi-layer structure of Kellomäki [Kel14]. The VP method uses a 2D simulation grid divided into 2D cells. In turn, each cell has one or multiple columns when we extend to the multi-layered VP model. Each column corresponds to a range $[\min_i, \max_i]$ along the vertical axis, with base height b_i, liquid height h_i, and maximum range $\max_i$ (Fig. 2). Adjacent columns are

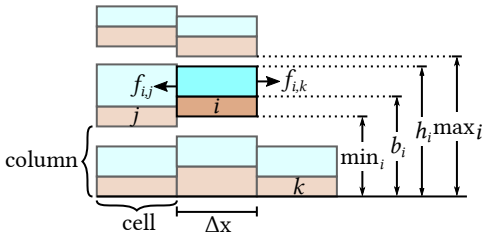


Figure 2: *The simulation grid is divided into cells, which in turn contain one or more columns. Here we see three linked columns i, j, and k, and column i's properties: its range $[min_i, max_i]$ along the vertical axis, its base height b_i, and its liquid height h_i.*

connected using virtual pipes through which they exchange liquid. In the multi-layer case, a pipe connects two adjacent columns i and j when their liquid ranges $[b_i, \max_i]$ overlap. Note that one column can be connected to multiple columns from the same adjacent cell. As in the work of Mei et al. [MDH07], pipe connections are created only for the 4-neighborhood. These connections are created at the beginning of the simulation and updated when there is a change in the geometry of the obstacles. Throughout this paper, we often use the term *terrain* to refer to any obstacle (soft or rigid) lying below the columns of liquid. At each simulation step, the flux $f_{i,j}$ between a column i and its neighbor column j is updated using their difference in hydrostatic pressure and an explicit Euler integration:

$$f_{i,j}^{t+\Delta t} = \zeta f_{i,j}^{t} + \Delta t A \frac{g(h_i^t - h_j^t)}{l}, \tag{1}$$

where ζ is the friction parameter introduced by Mould and Yang [MY97], Δt is the simulation timestep, g is the gravitational acceleration, A is the cross-section area of the virtual pipe, and l is the length of the virtual pipe. As in the work of Št'ava et al. [vBBK08], we set l to be equal to the grid cell size Δx, and $A = \Delta x^2$. To remove the dependency of the value of ζ on the timestep, we propose to set $\zeta = \omega^{\Delta t}$, where $\omega = [0,1]$ is the fraction of flux retained per unit time. In our examples, we set $\omega = 0.5$. Note that $f_{i,j} = -f_{j,i}$; thus we compute the flux once per pipe. In order to prevent the liquid height from going below the base height, the outflow fluxes are scaled as described by Mei et al. [MDH07]. Similarly, to prevent the liquid height from going above the maximum height, we scale the inflow fluxes as explained by Kellomäki [Kel14]. The scaled fluxes are then used to compute the new liquid heights, again using an explicit Euler integration:

$$h_i^{t+\Delta t} = h_i^t + \frac{\Delta t}{\Delta x^2} \sum_{j \in \text{neighbor}(i)} f_{j,i}^{t+\Delta t}.$$

Each frame, the fluxes are first computed, and then the liquid heights are updated using the new fluxes. The main steps of our simulation loop are:

1. **Update extended pipes connections (Sec. 3.3)**
2. Update liquid fluxes
3. **Apply viscosity (Sec. 3.2)**
4. Scale liquid fluxes
5. Update liquid heights
6. Source liquid in the simulation

Lines in bold indicate the steps that we added to the VP method to handle viscosity and the flow under fully-flooded passages.

3.2. Viscosity model

When a thin layer of liquid flows on a surface, it is slowed down by the shear stress forces from its interaction with the terrain. This effect is propagated further away through the liquid by the viscous shear forces. As the depth of the liquid layer increases, the impact of this interaction on the overall liquid velocity decreases, at a rate determined by its viscosity. Handling a wide range of liquid depths is important in may applications, including virtual surgery, for example. Hence, our goal is to handle a wide range of depths in a physically-based manner, which cannot be done by the current VP method.

Our proposed viscosity model is based on the Navier-Stokes equations, which are based on the velocity. The VP method works with the flux $f_{i,j}$ instead of the velocity $u_{i,j}$; These are related as follows:

$$f_{i,j} = \frac{u_{i,j}}{C}, \tag{2}$$

where $C = \Delta x(h-b)$ is the cross-sectional area of the liquid's flow from one column to its neighbor. Note that $u_{i,j}$ is a scalar that represents the speed of the flow along the pipe orientation. For simplicity

we assume that the vertical axis is z; pipes are thus aligned with either the x- or y- axis. Because $u_{i,j}$ lies between adjacent columns, it can be interpreted as a staggered grid, where $u_{i,j} = u_x$ or u_y, depending on its orientation.

The VP method assumes that the liquid's properties are constant along the vertical axis. However, because viscous forces are computed from the spatial differences of the velocity, this assumption would result in no shear viscous forces along the vertical axis. For that reason, we define a velocity profile that varies vertically $u^p_{i,j}(z)$, and interpret $u_{i,j}$ as the average vertical velocity. We derive the relationship between $u^p_{i,j}(z)$ and $u_{i,j}$ in App. A, which leads to:

$$u^p_{i,j}(z) = -\frac{3u_{i,j}}{H^2}\left(\frac{z^2}{2} - Hz\right), \tag{3}$$

where H is the liquid's depth $(h_i - b_i)$ of the column from which the flux originates. Using this vertically varying velocity, we derive (App. B) the effect of the viscosity on the average velocity $u_{i,j}$:

$$u^{n+1}_{i,j} = \left(\frac{H^2}{H^2 + 3\Delta t\nu}\right) u^n_{i,j},$$

where ν is the kinematic viscosity of the liquid. Using Eq. 2, we can convert from velocity to flux:

$$f^{n+1}_{i,j} C^{n+1} = \left(\frac{H^2}{H^2 + 3\Delta t\nu}\right) f^n_{i,j} C^n.$$

By assuming that the liquid depth remains the same during the timestep, we get $C^n = C^{n+1}$:

$$f^{n+1}_{i,j} = \left(\frac{H^2}{H^2 + 3\Delta t\nu}\right) f^n_{i,j}. \tag{4}$$

Each step, this equation is used to enforce viscosity on the fluxes computed from Eq. 1.

In Eq. 4, the right-hand side coefficient is guaranteed to be in the range $[0,1]$ for $0 \leq 3\Delta t\nu$ and $0 < H$. This means that no energy is added in the simulation, and thus it remains stable even for arbitrarily small or large kinematic viscosity values. In terms of behavior, as the liquid depth H increases, the coefficient of the right-hand side gets closer to 1, lessening the impact of viscosity. Additionally, the viscosity ν damps the fluxes with a greater impact as H becomes smaller. This shows that our viscosity model is stable and produces the intended behavior. Furthermore, it is simple and fast to compute, making it ideal for real-time purposes. Deriving this simple, stable, and efficient model required some simplifications from the Navier-Stokes equations. One notable simplification is that we neglect the contribution from neighbor cells to the viscosity (App. B). While our model does not capture all the features of the complete viscosity model, we will see in Sec. 6 that our animations live up to the expected behavior of a viscous fluid.

3.3. Multi-layer and extended pipes

The multi-layer VP method can only exchange liquid among neighbor columns. If there is a passage through or below obstacles, then as soon as the level of one column of the passage (Fig. 3 left, p_{in}) reaches its maximum, the VP method will block the flow of liquid.

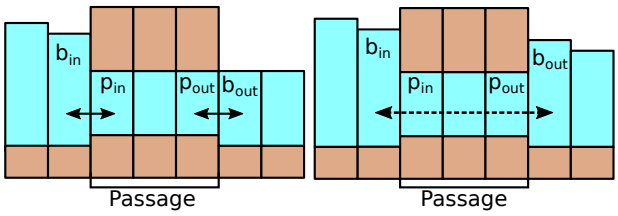


Figure 3: *Left: The VP method prevents flow between b_{in} and b_{out} (it is blocked at p_{in}). Right: Our approach permits flow between b_{in} and b_{out} using an extended pipe.*

To allow liquid flow inside a passage, we identify fully-flooded passages and change the connections using extended pipes to link both ends of the passage (Fig. 3, right). Thus, instead of having only connections among neighboring columns, we will also connect the ends of fully-flooded passages. A fully-flooded passage consists of a group of consecutive columns with pipe connections having a level of liquid equal to their maximum limit. For efficiency reasons, we restrict the search for consecutive cells along two orientations, namely, alignment with the local x- and y- axes of the grid. Once a fully-flooded passage is identified, we use an extended pipe to make a connection between the columns at the boundary (b_{in} and b_{out}). The flux between these columns is initialized to zero. Once we have adjusted the pipe connections of the columns on both sides of the passage, the standard VP method is used with the new connections. At each timestep, extended pipes are connected and disconnected as needed based on the identified fully-flooded passages.

4. Liquid surface

There are several requirements for the generation of the liquid surface: real-time generation and rendering, support for multi-layer surfaces, and flexibility to use an arbitrary geometry for the obstacles' surface. To render the liquid surface, Borgeat et al. [BMPG11] displace the terrain using the liquid height in the nearest column, and adjust the vertex colors and normals. As such, the following constraints are imposed on the meshes of the obstacles: the need to have the appropriate resolution and uniformity to correctly represent the liquid. Furthermore, discontinuities are introduced at overhangs where the obstacle meshes of both levels are not connected to each other (see accompanying video). Kellomäki [Kel14] displaces the vertices of a regular mesh grid based on the height of the topmost columns, but this method assumes a single continuous surface over the whole simulation domain, which is not the case in multi-layer scenarios. Considering the limitations of current methods, we devised a new surface creation approach which detects links between adjacent columns (Sec. 4.1), and connects them using an optimal triangle configuration (Sec. 4.2). Furthermore, we handle boundaries of the liquid to ensure a coherent surface (Sec. 4.3). Finally, we propose an optimization approach to prevent conflicting intersections between the liquid surface and the surface of the obstacles (Sec. 4.4).

4.1. Multi-layered surface links

When handling multiple layers, the surface of the liquid gets more complex: adjacent cells could have a different number of columns, and parts of the obstacle geometry can exhibit overhangs. This increases the complexity of the surface creation. It is thus important to derive a robust approach that can handle all cases. In our approach, neighbor columns that will be linked by the liquid mesh surface are identified using a multi-layered link test (Fig. 4). With this test, columns are linked if their respective liquid heights fit between each other's minimum and maximum heights. As such, we define two neighbor columns i and j as linked if: $\min_j < h_i < \max_j$, and $\min_i < h_j < \max_i$. In contrast to the pipe connections, the surface links are computed for the 8-neighborhood of each cell. They are computed for each pair of adjacent wet or wet–dry columns; a dry column is defined as a column where $h_i \leq b_i$. This approach works in the general case, while a few special cases occurring at the boundary are handled differently (Sec. 4.3). The surface links change over the course of the simulation and are updated at each frame. In the next section, we explain our approach to generating the surface from the surface links.

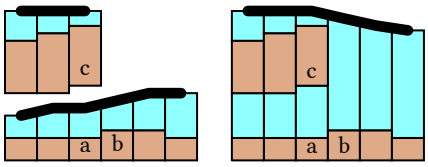


Figure 4: *Multi-layered linking. The thick lines show linked columns. On the left: columns* a *and* b *are initially linked. On the right: as the liquid height of* b *increases, it becomes linked with* c.

4.2. Surface creation

Our work extends the idea of displacing the vertices of a regular mesh grid to multi-layered simulations. At the beginning of the simulation, a 3D vertex is allocated for each column, and matches the 2D coordinates of its column. Then, at each frame, its vertical position is updated to match the liquid's height. Afterward, these vertices are used to create triangular faces among adjacent linked columns (based on the link test from Sec. 4.1). To that end, quartets of neighbor cells $\{(i,j),(i+1,j),(i,j+1),(i+1,j+1)\}$ are iterated. For each quartet of cells, we analyze the links among their columns. First, we find the groups of four mutually interlinked columns (these will form two triangles). From the remaining columns, we then identify the groups of three mutually interlinked columns (these will form a single triangle). Finally, the remaining groups of two mutually interlinked columns are discarded as they do not correspond to a triangle. For groups of four linked neighbor columns, two triangle configurations are possible. In such cases, we pick the configuration for which the sum of the liquid height of the diagonal endpoints is the greatest, as described by Chentanez and Müller [CM10], in order to align the diagonal edge of the triangles with the curvature of the surface.

Before rendering, the normals are updated from the liquid heights, and the vertex opacity o_i is adjusted to the liquid depth:

$$o_i = \min\left(\frac{h_i - b_i}{\text{depth}_{\max}}, 1\right),$$

where $\text{depth}_{\max}$ is the depth from which the liquid starts being opaque. This enhances the realism by better matching the opacity of the simulated liquid, and it improves the look of the wet–dry boundary on flat and convex surfaces.

4.3. Boundaries

In order to render a smooth liquid boundary and avoid having cracks at the junction with the terrain, the surface requires special treatment at the boundary of the liquid. Wet columns can be linked with dry columns having a base height (and thus liquid height) significantly higher than the liquid height of the wet column. Blindly creating triangles to the liquid height of dry columns would result in unrealistic slopes on the liquid surface near dry columns (Fig. 5 left). The vertex height of dry columns is thus adjusted to the average height of their linked wet neighbors, and the surface normals of dry columns are adjusted in a similar fashion. With these corrected vertices and normals, the surface is smoother and appears as expected (Fig. 5, right). Also, during the triangle generation step, the triangle configuration that allows the ends of the diagonal edge to be both inside or both outside the boundary is prioritized.

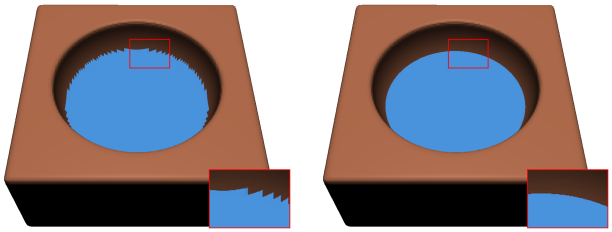


Figure 5: *A bowl filled with liquid without adjusting boundaries (left), and with boundary adjustments (right)*

In some cases, generally below overhangs, the liquid surface in a wet column i might be next to a wall boundary belonging to a neighbor column j to which it is not linked, i.e., $min_j < h_i < max_j$, but $max_i < b_j$. This results in a crack between the wet column and the wall boundary. To prevent such cracks, we create an extra vertex positioned at the center of column j. This new vertex is identified as linked to column i, as well as linked to its wet neighbor columns that are linked with column i. Its height and normal are then set to the average of these linked neighbor columns. The triangle generation step is triggered again for all such new boundary links.

4.4. Surface optimization for rendering

Because the rendered terrain geometry could come from an arbitrary source of data, it can be misaligned with the simulation grid. This often results in incorrect interpenetrations with the mesh of the liquid surface (Fig. 6 left). To prevent such issues, columns with thin layers of liquid are corrected: to prevent surface interpenetrations and z-fighting, any non-zero liquid height cannot go below

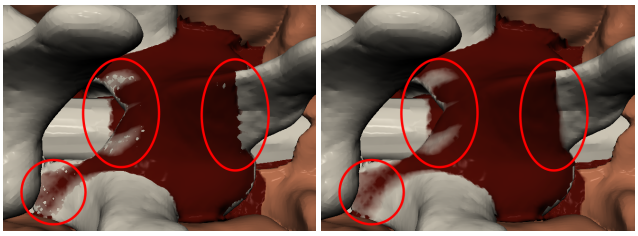


Figure 6: *A comparison without (left) and with (right) our minimum height correction of the liquid surface heights*

a computed minimum height (Fig. 6 right). This minimum height $h_k^{\min}$ is a combination of a global parameter to avoid z-fighting, and a locally-computed height to avoid interpenetrations. The global parameter is set to $0.05\Delta x$ in our examples. By contrast, the locally-computed height is different for each column: it is precomputed from the terrain mesh, and updated when the terrain changes.

The goal of the local height optimization is to compute a minimum height for each column to ensure that all triangles of the liquid surface will be above the terrain mesh vertices, even if the amount of liquid in a column is very small. We iterate through each vertex of the terrain mesh surface, identifying the quartet of four columns surrounding the vertex, and optimizing the local heights to make them as small as possible, under the constraint that the triangles formed by these columns are above the terrain vertex. We express the height of the liquid surface, h_l, along the vertical axis through a terrain vertex as the bilinear interpolation of the heights of the related quartet. The height of each column is the sum of the local height l_k and the base height b_k. We optimize the local heights l_k under the constraint that the bilinear interpolation h_l should be slightly above the terrain vertex h_v:

$$\min_{l_k} \sum_k \left(w_k l_k^2 \right), \quad \text{subject to} \quad h_v + \varepsilon = \underset{k \in \text{quartet}}{\text{bilinear}}(b_k + l_k), \tag{5}$$

where w_k are weights assigned to each column, which will be discussed later in this section. This constrained equation is solved analytically using the Lagrange multipliers. As such, the approach is efficient as it does not require that a system of equations be solved for each terrain vertex. The final l_k for a column is the maximum over the l_k computed for all terrain vertices. Once the local heights are computed, the local minimum height $h_k^{\min}$ of each of the four columns can be computed as $h_k^{\min} = b_k + l_k$.

The solution of Eq. 5 can create unnatural bumps on the surface, generally in regions near a steep slope on the terrain, where the differences among base heights in the quartet are quite large. We prevented such issues by designing weights w_k based on the distance between the column base height b_k and the vertex height h_v:

$$w_k = \begin{cases} \dfrac{\Delta x}{h_v - b_k}, & \text{if } b_k \leq h_v \\ 1\text{E}10, & \textit{otherwise} \end{cases}. \tag{6}$$

These weights enforce a greater correction to the columns with a base height at the bottom of a large slope, and limits corrections to those already above the terrain vertex height.

Terrain vertices whose normal is facing downward are skipped. Furthermore, we discard terrain vertices that create a height $h_k^{\min}$ higher than a cell size Δx above the highest base height of the quartet. Such cases usually generate unrealistic bumps near edges of large slopes. Finally, large corrections can be induced by terrain vertices that are extremely close to one of the four columns. To prevent such large corrections, we take the global minimum into account during the calculations of the local minimum height, constraining the l_k to be greater than or equal to the global minimum height.

5. Meniscus shading

The capillary action at the fluid–solid interface causes the fluid surface to curve near the borders, which affects the surface normals and results in specular highlights at the boundary. We rely on an inexpensive correction of the normal to provide the desired visual effect, similar to bump mapping (Fig. 7). First, we identify the

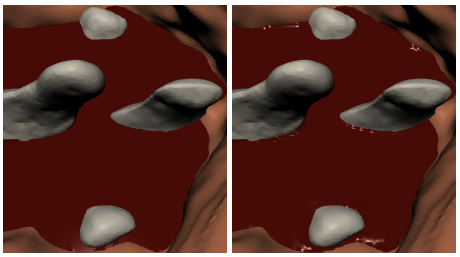


Figure 7: *An example of a rendering without meniscus (on the left), and with our approach (on the right)*

columns of liquid affected by the meniscus. This is done by computing a distance field to the boundary on the surface of the liquid, within the given meniscus range. For each meniscus column, we also obtain the direction to the boundary, as well as an estimate of the terrain inclination at the boundary. This information is needed for our last step, where we tilt the normals of the meniscus columns to emulate the expected curvature, as depicted in Fig. 8.

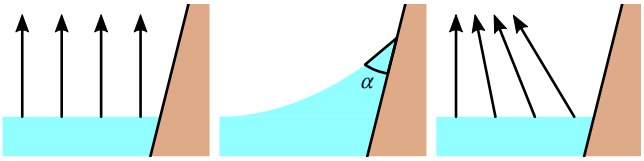


Figure 8: *Correction of the normals on the liquid surface based on the distance to the boundary (left: original normals, middle: expected curvature, right: corrected normals)*

5.1. Distance and direction to boundary

To identify the region containing the meniscus, we use the 2D distance and direction to the Nearest Boundary Column (NBC), i.e., the closest column on the "dry" side of the fluid–solid boundary. We proceed in a flood-fill manner from the NBC inward to the wet columns. We start from the *first ring* of wet columns, i.e., wet

columns linked to a dry column or an extra vertex below an overhang (Sec. 4.3). We use these links to initialize the NBCs and distances of the first ring. We then do several iterations to propagate the information from the boundary. The number of iterations is automatically set to cover the meniscus region based on the meniscus length and the cell size. During the iterations, for the current wet column, we consider its linked columns and use the Dead Reckoning algorithm [Gre04], a variant of the Chamfer distance transform algorithm: if the NBC distance of the current column is greater than that of a linked column summed with the distance to that column, we update the current NBC and distance (with the Euclidean distance to the NBC).

We also need the direction to the NBC for each column in the meniscus region in order to orient the meniscus accordingly. To that end, we can either compute the directions to the NBCs and then filter to attenuate any aliasing from the discrete grid distance field or use the finite central difference. We choose the latter, as it gives a smoother approximation than the raw directions, while limiting the amount of computation required.

5.2. Normal correction

The contact angle α between a liquid and a solid is given by Young's equation [You05]. The normal correction angle at the contact point (i.e., the fluid–solid interface) varies with the slope of the terrain at that point, as illustrated in Fig. 9. We use the base heights

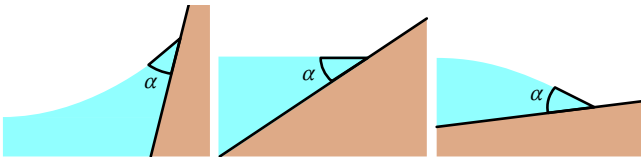


Figure 9: *The contact angle α affects the surface differently based on the inclination of the solid, resulting in a concave (left) or convex (right) meniscus, or no meniscus (middle)*

of our column and of its NBC to obtain an estimate of the tilt angle of the solid β. This allows us to compute the normal correction angle $\psi = \beta - \alpha$ at the contact point (Fig. 10).

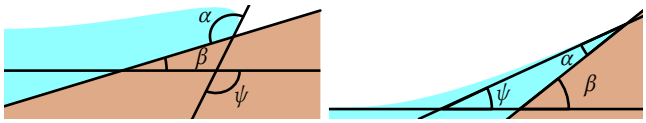


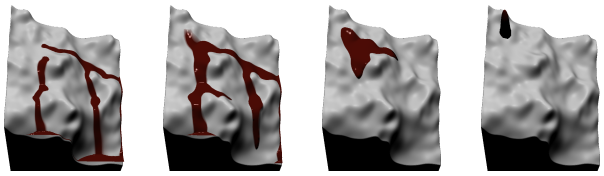
Figure 10: *From contact angle α and solid tilt angle β, we obtain the correction angle ψ to apply to the normal at the contact point*

We adjust the normals of the liquid surface for all the columns within the meniscus region. In our examples, we use a meniscus length of 2.8 mm, based on the densities of water and air [MWE16]. Using the distance to the NBC and the correction angle at the contact point, we find the tilting angle for each affected column, linearly interpolated from 0 at the maximal meniscus distance to the angle ψ at the interface. Finally, we rotate the normal vector accordingly, as shown in Fig. 8, using the horizontal vector perpendicular to the NBC direction. This approach results in a convincing shading effect that can emulate both convex and concave menisci.

6. Results

We tested our approach with a wide range of scenarios to show its behavior, as well as the impact of its parameters. Results are also available in the accompanying video. Parameter values as well as timings can be found in Table 1. As with typical fluid simulation methods with an explicit integration, the timestep should be adjusted according to the CFL condition. Viscosity values in the range $[1.0E-6, 1.0E-5]$ provide realistic results for most liquids such as water, blood, and paint.

An important contribution of our work is the viscosity model for small-scale liquid simulation (Sec. 3.2). We tested our approach with varying viscosity values (see the video and stills in Fig. 11). The images show the simulation at time $t = 3.0s$, with $\nu = 0$, 4.0E−6, 4.0E−5, and 4.0E−1 m^2/s. As the viscosity increases, the liquid flows more slowly on the inclined terrain. Even with a very large viscosity (Fig. 11(d)) our approach is stable.



(a) $\nu = 0$ **(b)** $\nu = 4.0E{-}6$ **(c)** $\nu = 4.0E{-}5$ **(d)** $\nu = 4.0E{-}1$

Figure 11: *A liquid flows on an inclined terrain with varying viscosity, ν, expressed in m^2/s*

We make several contributions to the optimization of the liquid surface. As shown in Sec. 4, we improve the smoothness of the boundaries, and we prevented incorrect interpenetrations. To demonstrate the surface construction with multiple layers, we show a three-layer scenario with cavities and overhangs in the accompanying video and in Fig. 12. Through the simulation, the surfaces of

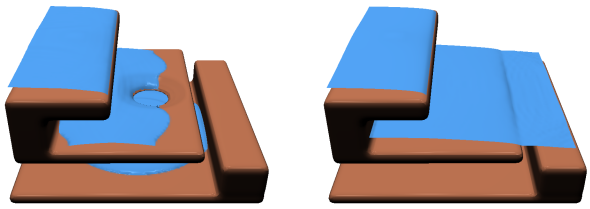


Figure 12: *This scenario shows how our surface linking approach correctly handles multiple changes in the surface topology*

the multiple layers link the appropriate columns, even with this configuration exhibiting overhangs and a hole in the middle platform. We also validated extended pipes on a scenario with four passages. Fig. 13 shows how, after the passages are flooded, the liquid transfer does not stop, and is able to fill the other parts of the container.

		Δt	Δx	ν	Simulation		Surface		Rendering		Total	
Example	**Resolution**	**(ms)**	**(m)**	**(m^2/s)**	**Max**	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**	**Avg**
Surgery (Fig. 1)	$200 \times 200 \times 5$	3.00	0.0005	4.0E−6	5.43	4.86	2.11	1.10	0.66	0.34	7.04	6.30
Viscosity (Fig. 11(a))	$200 \times 200 \times 1$	3.00	0.0005	0	1.05	0.61	0.49	0.43	0.19	0.16	1.63	1.21
Viscosity (Fig. 11(b))	$200 \times 200 \times 1$	3.00	0.0005	4.0E−6	1.03	0.62	0.52	0.46	0.21	0.16	1.63	1.22
Viscosity (Fig. 11(c))	$200 \times 200 \times 1$	3.00	0.0005	4.0E−5	1.02	0.60	0.52	0.44	0.25	0.16	2.41	1.21
Viscosity (Fig. 11(d))	$200 \times 200 \times 1$	3.00	0.0005	4.0E−1	0.93	0.60	0.52	0.41	0.22	0.16	2.40	1.20
Three layers (Fig. 12)	$100 \times 100 \times 3$	9.00	0.001	4.0E−6	0.68	0.32	0.85	0.37	0.89	0.17	1.53	0.86
Passages (Fig. 13)	$100 \times 100 \times 3$	9.00	0.001	4.0E−6	0.50	0.26	0.48	0.34	1.10	0.15	2.36	0.78

Table 1: *Statistics for all the examples shown in this paper: resolution ($N \times N \times Layers$), timestep Δt, cell size Δx, and kinematic viscosity ν as well as maximum/average timings (in ms) per frame for the simulation, surface generation, and rendering*

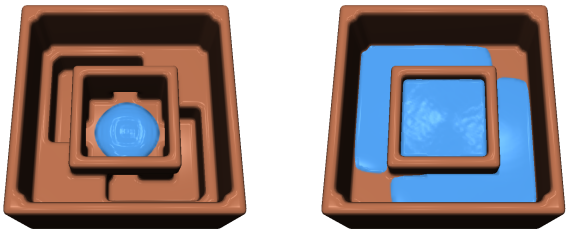


Figure 13: *The liquid flows from the center and through four passages (left). When the passages are fully flooded, the extended pipes continue the transfer of liquid (right).*

We also validated our approach with a surgery simulation scenario, including multiple overhangs and holes. In Fig. 1, blood originates on top of a vertebra during surgery. As the simulation domain is filled, our surface correctly handles the evolving topology of the liquid surface, and no holes or discontinuities are visible.

We executed the tests presented in this paper on an Intel Core i5-6600 3.30 GHz with 16 GB of RAM and a GeForce GTX 970. To take advantage of the GPU parallelism, our approach is implemented using CUDA. The timings (Table 1) show that all of our examples run in real time. The most computationally expensive part is the simulation step which takes around 50-80% of the total computation time in our examples. All of our examples take considerably less than the 16.6 ms needed to achieve real-time rates, leaving time for other computations such as soft body simulation, haptic feedback, rendering, and collision detection.

7. Discussion

We show in the accompanying video that our viscosity model, surface reconstruction, and meniscus modeling are independent of the underlying height map simulation; it works just as well with the Shallow Water Equation (SWE) simulation [CM10]. While the SWE method can take advantage of our contributions, the VP method proved to be a better choice since it had results of equivalent quality with significantly smaller computation times.

The simulation behavior is improved by the use of extended pipes, which allow the liquid to flow through fully-flooded passages. However, the extended pipes approach sometimes introduces some ripples when these pipes get connected and disconnected, even when using the scaling method of Mei et al. [MDH07].

The surface constructed by our approach can handle the evolving topology of the liquid throughout the simulation. However, there is no geometry created to close the gap near edges between unlinked cells (e.g., the edge of an overhang). Nonetheless, these are in areas where the liquid *falls* down, thus the water height is generally low, and the gap is not very apparent. Another downside is that the grid generally does not follow exactly the silhouette at the edge of these overhangs, which sometimes results in having a part of the geometry that will never be covered by the liquid surface. Additionally, the dry/wet boundary moves on a cell-by-cell basis, which can result in some popping. While the depth-based opacity significantly reduces this issue, it is still visible in our examples. This behavior can be improved by increasing the resolution of the grid, or by tracking the surface as suggested by Thuerey and Hess [MSJT08, Chapter 11].

In our approach, we modify only the normals and not the mesh. This may not trigger enough fragments to give an acceptable highlight, especially when viewing the surface of the fluid from the side. However, modifying the mesh would be more demanding as we would need to handle problems involving cracks and interpenetrations between the surface of the meniscus and the solid.

8. Conclusion

In this paper, we have presented a real-time shallow water simulation approach for small-scale scenarios. It introduces a fast and stable viscosity model that can be used for any heightmap simulation method, such as the VP or the shallow water equations. It improves the behavior of multi-layered simulations by handling the flow inside passages. Moreover, it constructs a triangular mesh surface that accounts for the interlinks among the multiple layers of liquid. We also introduce a new surface optimization approach that computes minimum heights of the liquid surface in order to prevent interpenetration of the underlying terrain surface. Finally, we improved the shading with a novel real-time meniscus approach that proved to be useful for the VP, but can also be used for other heightmap-based simulation methods. All these contributions improve the realism

of small-scale simulations. Furthermore, the approach is computationally inexpensive, which allows it to co-exist with other real-time systems normally required in the context of real applications such as surgery training and games. We have demonstrated that the approach works in real time for various scenarios such as blood simulation for virtual surgery.

Although we improved the surface boundaries while maintaining real-time performance, the former could be further improved by closing the gap near overhanging edges. In those regions, the liquid falling from one layer to another could be simulated using particles to improve realism, such as in the work of Chentanez and Müller [CM10]. We restricted ourselves to linear passages to be computationally efficient. In future works, we could expand the variety of passages that can be handled, but doing so while balancing realism and efficiency is not an easy task. Finally, we believe the computation time for the meniscus shading could be improved by using a precomputed distance field for static solids such as in the work of Morgenroth et al. [MWE16].

Acknowledgements

The work was supported by OSSimTech, Digital District, NSERC, Mitacs, and Prompt. We would like to thank Side Effects Software for providing Houdini licenses.

Appendix A: Velocity profile

In this section, we derive the vertically varying velocity $u^p_{i,j}(z)$. For simplicity, we assume that the base of the liquid is at $z = 0$ and that the top is at $z = H$. The function $u^p_{i,j}(z)$ is thus defined inside the liquid, i.e., for $0 \leq z \leq H$. The velocity is derived from the incompressible Navier-Stokes equations:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} - \nu \nabla \cdot \nabla \vec{u} = -\frac{1}{\rho} \nabla p + \vec{g}, \tag{7}$$

$$\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} = 0, \tag{8}$$

where p is the pressure, $\vec{g} = (0, 0, g_z)$ is the gravity, and $\vec{u} = (u_x, u_y, u_z)$ is the velocity. To make the velocity profile a function of only the height z inside the liquid, we show that the Navier-Stokes momentum equation (Eq. 7) can be simplified. To begin, the first term can be removed by considering a steady flow:

$$\frac{\partial \vec{u}}{\partial t} = 0. \tag{9}$$

We also assume the velocity is constant along the horizontal plane:

$$\frac{\partial \vec{u}}{\partial x} = \frac{\partial \vec{u}}{\partial y} = \vec{0}, \tag{10}$$

Using this assumption, Eq. 8 yields:

$$\frac{\partial u_z}{\partial z} = 0. \tag{11}$$

At the terrain level, we use a no-slip boundary condition, and at the surface level a traction-free boundary condition:

$$\vec{u}|_{z=0} = \vec{0}, \; \frac{\partial \vec{u}}{\partial z}\bigg|_{z=H} = \vec{0}. \tag{12}$$

The terrain boundary condition and Eq. 11 tell us that $u_z = 0$ everywhere, and can thus be ignored. Using this result and Eq. 10, the second term of the Navier-stokes momentum equation becomes zero:

$$\vec{u} \cdot \nabla \vec{u} = u_x \frac{\partial \vec{u}}{\partial x} + u_y \frac{\partial \vec{u}}{\partial y} + u_z \frac{\partial \vec{u}}{\partial z} \tag{13}$$

$$= u_x(0) + u_y(0) + (0) \frac{\partial \vec{u}}{\partial z} = \vec{0}. \tag{14}$$

Finally, the third term can be simplified similarly:

$$-\nu \nabla \cdot \nabla \vec{u} = -\nu \left(\frac{\partial^2 \vec{u}}{\partial x^2} + \frac{\partial^2 \vec{u}}{\partial y^2} + \frac{\partial^2 \vec{u}}{\partial z^2} \right). \tag{15}$$

Using Eq. 10, the first two terms become zero, yielding:

$$-\nu \nabla \cdot \nabla \vec{u} = -\nu \frac{\partial^2 \vec{u}}{\partial z^2}. \tag{16}$$

After these simplifications, Eq. 7 now becomes:

$$-\nu \frac{\partial^2 \vec{u}}{\partial z^2} = -\frac{1}{\rho} \nabla p + \vec{g}. \tag{17}$$

Now that the Navier-Stokes equations have been simplified, we can derive the horizontal velocity as a function of z. As we use fluxes between cells, we derive the velocity along u_x- and u_y- axes. We will show u_x, and u_y can be derived in the same way. From Eq. 17, considering the x component, we get:

$$\nu \frac{\partial^2 u_x}{\partial z^2} = \frac{1}{\rho} \frac{\partial p}{\partial x}. \tag{18}$$

We assume the horizontal pressure variation to be constant, and replace it by a constant w:

$$\nu \frac{\partial^2 u_x}{\partial z^2} = w, \tag{19}$$

Integrating both sides twice with respect to z, using the boundary conditions from Eq. 12 to find the values of the integration constants, dividing both sides by ν, and rearranging the terms yields:

$$u_x = \frac{w}{\nu} \left(\frac{z^2}{2} - Hz \right). \tag{20}$$

To link the velocity function to the flux velocity $u_{i,j}$, we find the expression of the pressure gradient w that will make the average velocity of u_x inside the liquid equal to $u_{i,j}$

$$u_{i,j} = \frac{1}{H} \int_0^H \frac{w}{\nu} \left(\frac{z^2}{2} - Hz \right) dz = -\frac{wH^2}{3\nu} \tag{21}$$

$$w = -\frac{3\nu u_{i,j}}{H^2} \tag{22}$$

Using that result in Eq. 20 yields:

$$u_x = -\frac{3u_{i,j}}{H^2} \left(\frac{z^2}{2} - Hz \right) \tag{23}$$

Performing the same derivation on u_y yields the same result. Knowing that a flux is always aligned with the x or y axis, u_x can be substituted by $u^p_{i,j}(z)$:

$$u^p_{i,j}(z) = -\frac{3u_{i,j}}{H^2} \left(\frac{z^2}{2} - Hz \right) \tag{24}$$

Appendix B: Viscosity model

In this section, we derive the viscosity model, using the vertically varying velocity $u^p_{i,j}(z)$ (App. A). Our goal is to determine how to update the velocity, i.e., $\frac{\partial u_{i,j}}{\partial t}$, based on the viscosity. Our model accounts for the effect of viscosity from the terrain to the liquid surface, ignoring the effect of neighbor cells. While this is less accurate, neglecting the relation between neighbor cells allows us to implicitly integrate each column independently, instead of having to solve a large system of equations. We derive our viscosity model by considering only the viscous term of the Navier-Stokes momentum equation. Here, we show the derivation for $\frac{\partial u_x}{\partial t}$, but the same process can be applied to $\frac{\partial u_y}{\partial t}$:

$$\frac{\partial u_x}{\partial t} = \nu \left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} + \frac{\partial^2 u_x}{\partial z^2} \right). \tag{25}$$

With the assumption of Eq. 10, Eq. 25 is simplified to:

$$\frac{\partial u_x}{\partial t} = \nu \frac{\partial^2 u_x}{\partial z^2} \tag{26}$$

Using the velocity profile defined by Eq. 24, this equation becomes:

$$\frac{\partial u_x}{\partial t} = \nu \frac{\partial^2}{\partial z^2} \left[-\frac{3u_{i,j}}{H^2} \left(\frac{z^2}{2} - Hz \right) \right] = -\frac{3\nu u_{i,j}}{H^2} \tag{27}$$

This result is used to compute how the velocity $u_{i,j}$ varies over time:

$$\frac{\partial u_{i,j}}{\partial t} = \frac{\partial}{\partial t} \left(\frac{1}{H} \int_0^H u_x dz \right). \tag{28}$$

Using the Leibniz integral rule, and simplifying the problem by assuming that the liquid depth H does not vary over time, yields:

$$\frac{\partial u_{i,j}}{\partial t} = \frac{1}{H} \int_0^H \left(-\frac{3\nu u_{i,j}}{H^2} \right) dz = -\frac{3\nu u_{i,j}}{H^2}. \tag{29}$$

Finally, we integrate the average velocity implicitly using the backward Euler method to get the temporal update of the velocity considering the viscosity:

$$u^{n+1}_{i,j} = u^n_{i,j} - \frac{3\Delta t \nu}{H^2} u^{n+1}_{i,j}. \tag{30}$$

Rearranging the terms yields:

$$u^{n+1}_{i,j} = \left(\frac{H^2}{H^2 + 3\Delta t \nu} \right) u^n_{i,j}. \tag{31}$$

References

[AGL*17] AANJANEYA M., GAO M., LIU H., BATTY C., SIFAKIS E.: Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Trans. Graph. 36*, 4 (July 2017), 140:1–140:12. 2

[ATBG08] ANGST R., THUEREY N., BOTSCH M., GROSS M.: Robust and efficient wave simulations on deforming meshes. *Computer Graphics Forum 27*, 7 (2008), 1895–1900. 2

[BMPG11] BORGEAT L., MASSICOTTE P., POIRIER G., GODIN G.: Layered surface fluid simulation for surgical training. In *Proc. of Medical Image Computing and Computer-Assisted Intervention – MICCAI 2011 Conference* (2011), Springer, pp. 323–330. 2, 4

[CM10] CHENTANEZ N., MÜLLER M.: Real-time simulation of large bodies of water with small scale details. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comp. Animation* (2010), SCA '10, pp. 197–206. 1, 2, 5, 8, 9

[EMF02] ENRIGHT D., MARSCHNER S., FEDKIW R.: Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3 (July 2002), 736–744. 2

[FAW*16] FERSTL F., ANDO R., WOJTAN C., WESTERMANN R., THUEREY N.: Narrow band flip for liquid simulations. *Computer Graphics Forum 35*, 2 (2016), 225–232. 2

[Gre04] GREVERA G. J.: The "dead reckoning" signed distance transform. *Computer Vision and Image Understanding 95*, 3 (2004), 317–333. 7

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: SPH Fluids in Computer Graphics. In *Eurographics 2014 - State of the Art Reports* (2014), Lefebvre S., Spagnuolo M., (Eds.), The Eurographics Association. 2

[JSS*15] JIANG C., SCHROEDER C., SELLE A., TERAN J., STOMAKHIN A.: The affine particle-in-cell method. *ACM Trans. Graph. 34*, 4 (July 2015), 51:1–51:10. 2

[JW17] JESCHKE S., WOJTAN C.: Water wave packets. *ACM Trans. Graph. 36*, 4 (2017), 103:1–103:12. 2

[Kel14] KELLOMÄKI T.: Rigid body interaction for large-scale real-time water simulation. *Int. J. Comput. Games Technol.* (2014). 1, 2, 3, 4

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. *Comput. Graph. 24*, 4 (Sept. 1990), 49–57. 2

[KSS09] KERWIN T., SHEN H.-W., STREDNEY D.: Enhancing Realism of Wet Surfaces in Temporal Bone Surgical Simulation. *IEEE Trans. on Visualization and Computer Graphics 15*, 5 (Feb. 2009), 747–758. 2

[LO07] LEE R., O'SULLIVAN C.: A Fast and Compact Solver for the Shallow Water Equations. In *Workshop in Virtual Reality Interactions and Physical Simulation (VRIPHYS)* (2007), pp. 51–57. 2

[LvdP02] LAYTON A. T., VAN DE PANNE M.: A numerically efficient and stable algorithm for animating water waves. *The Visual Computer 18*, 1 (Feb 2002), 41–53. 2

[MDH07] MEI X., DECAUDIN P., HU B.-G.: Fast hydraulic erosion simulation and visualization on gpu. In *Pacific Conference on Computer Graphics and Applications, PG '07* (Oct 2007), pp. 47–56. 1, 3, 8

[MM13] MACKLIN M., MÜLLER M.: Position based fluids. *ACM Trans. Graph. 32*, 4 (July 2013), 104:1–104:12. 2

[MSJT08] MÜLLER M., STAM J., JAMES D., THÜREY N.: Real time physics. In *SIGGRAPH Class Notes* (2008), ACM, pp. 88:1–88:90. 8

[MWE16] MORGENROTH D., WEISKOPF D., EBERHARDT B.: Direct raytracing of a closed-form fluid meniscus. *The Visual Computer 32*, 6-8 (June 2016), 791–800. 7, 9

[MY97] MOULD D., YANG Y.-H.: Modeling water for computer graphics. *Computers & Graphics 21*, 6 (1997), 801–814. 3

[OH95] O'BRIEN J. F., HODGINS J. K.: Dynamic simulation of splashing fluids. In *Proc. of Computer Animation '95* (Apr 1995), pp. 198–205, 220. 2, 3

[vBBK08] ŠT'AVA O., BENEŠ B., BRISBIN M., KŘIVÁNEK J.: Interactive terrain modeling using hydraulic erosion. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.* (2008), SCA '08, Eurographics Association, pp. 201–210. 1, 3

[WMT07] WANG H., MILLER G., TURK G.: Solving general shallow wave equations on surfaces. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.* (2007), SCA '07, Eurographics Association, pp. 229–238. 2

[YHK07] YUKSEL C., HOUSE D. H., KEYSER J.: Wave particles. *ACM Trans. Graph. 26*, 3 (2007). 2

[You05] YOUNG T.: An essay on the cohesion of fluids. *Philosophical Trans. of the Royal Society of London 95* (1805), 65–87. 7