ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2020 / J. Bender and T. Popa (Guest Editors)

Volume 39 (2020), Number 8

Particle-based Liquid Control using Animation Templates

Arnaud Schoentgen [1,2] Pierre Poulin [1] Emmanuelle Darles [2] Philippe Meseure [2]

[1] Université de Montréal, Montréal, Canada [2] Université de Poitiers, Poitiers, France

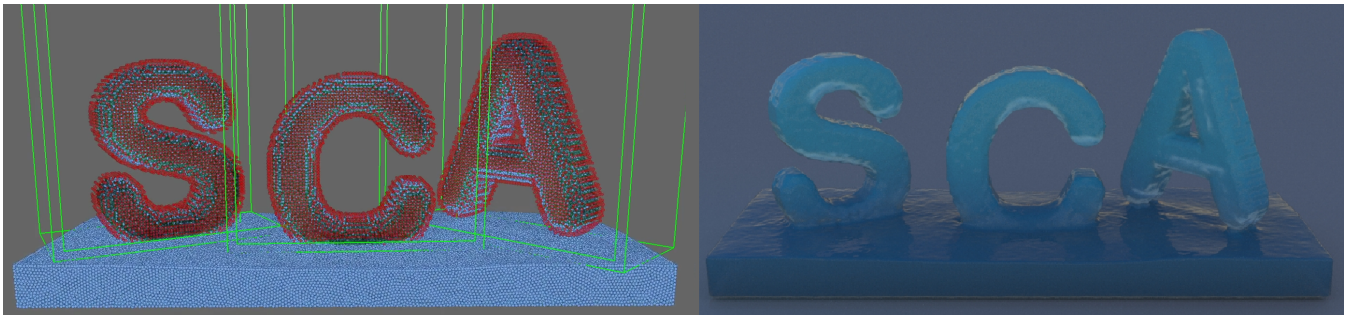

Figure 1: *Left: Three separate animation templates forming letters SCA are played in reverse order to generate localized forces such that particles emerge from the liquid below. Repulsion particles (in red) ensure an improved control of the liquid surface. Right: Rendering of the liquid surface.*

Abstract

It is notoriously difficult for artists to control liquids while generating plausible animations. We introduce a new liquid control tool that allows users to load, transform, and apply precomputed liquid simulation templates in a scene in order to control a particle-based simulation. Each template instance generates control forces that drive the global simulated liquid to locally reproduce the templated liquid behavior. Our system is augmented with a variable proportion of temporary particles to help efficiently reproduce the templated liquid density, with fewer requirements on the surrounding environment. The resulting control strategy adds only a small computational overhead, leading to quick visual feedback for resolutions allowing interactive simulation. We demonstrate the robustness and ease of use of our method on various examples in 2D and 3D.

CCS Concepts

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

1. Introduction

A physically-based animation can be very complex and realistic, but controlling its results is a daunting task. A simulation-based animation depends on many parameters, including its initial state, its time integration scheme, and the underlying physics. Fluid simulation brings up particularly difficult problems to the controlling process because of the complexity and nonlinearity of the Navier-Stokes equations. Many methods and tools have been developed to convincingly simulate fluids. However, too few methods provide efficient and intuitive ways to give a targeted behavior to a fluid. Ideally, an easy-to-use and intuitive tool should rely on known control metaphors. Moreover, when the resolution of the fluid allows for interactive simulation, a fluid editing method should be fast enough to provide quick visual feedback to the animator in order to avoid hampering the artistic process.

This paper introduces a particle-based liquid simulation and editing system that relies on the use of precomputed simulation templates stored in a database. Our system allows a user to efficiently compose liquid simulations using these animation templates, with a small computational overhead compared to the actual simulation. Once located and applied in both space and time, the template instances are used to generate control forces as well as temporary liquid particles in order to efficiently reproduce the animator's vision, with fewer requirements on the initial state and the surrounding environment of the simulation. The resulting animations are both plausible and predictable, as illustrated in Figure 1. In our system,

each template instance can be modified using a set of traditional animation-editing metaphors familiar to computer graphics (CG) artists, including spatial and temporal transformations. Our user interface is built on these metaphors.

More specifically, our contributions are:

- a novel, intuitive, and handy way for artists to generate both 2D and 3D liquid animations with an abstracted compositing tool;
- the formulation of spatial and temporal transformations over a precomputed template, with a surrounding falloff region;
- the use of a layer of repulsion particles to correctly reproduce a template liquid interface; and
- the introduction of temporary particles to quickly and efficiently reproduce a liquid template, with fewer requirements on the surrounding environment.

The rest of this paper is organized as follows. We first discuss related work on fluid control in Section 2. Our method is described in Section 3, and details of our implementation are provided in Section 4. Results are then presented and analyzed in Section 5, before conclusions and future challenges are drawn in Section 6.

2. Previous Work

Fluid control has been an active field of research in computer graphics for many years. In their original work, Foster and Metaxas [FM97] controlled fluids by modifying physical properties such as pression and surface tension.

Methods have been proposed to cast the fluid control problem as a space-time optimization over all possible control forces constrained by the Navier-Stokes equations. In these methods, a small set of sketched shapes at given keyframes are matched by applying forces found using a gradient-based optimizer [TMPS03, MTPS04]. To improve efficiency by up to an order of magnitude, Pan and Manocha [PM17b] split this optimization into two subproblems, solved with the alternating direction method of multipliers (ADMM). Even though their results are visually satisfying, these optimization methods are very far from delivering interactive results, even for low-resolution grids. Inglis et al. [IEGT17] minimize with a first-order Primal-Dual method the differences between a guided velocity field and a target velocity field. Their method correctly guides smoke through a target velocity field while still conserving turbulent effects. Pan et al. [PHT*13] locally edit low-resolution liquid simulations by solving an optimization problem on a subset of particles. However, an optimal control problem becomes expensive when applied to large control regions.

Using geometric considerations, Raveendran et al. [RWTT14] generate a plausible liquid animation by interpolating two precomputed liquid simulations. In order to perform data-driven fluid simulations, Thuerey [Thu16] combines fluid simulations with interpolation techniques that mimic the Navier-Stokes equations.

To control smoke simulations, Fattal and Lischinski [FL04] use two empirical forces: a driving force that guides smoke towards a target density field, and a smoke gathering force that prevents smoke from diffusing. Hong and Kim [HK04] extract a potential field from a static user-defined target during preprocessing, before applying a control force to a smoke simulation, proportional to the gradient of this field. Shi and Yu apply empirical feedback forces to make a controlled fluid match a given target in both smoke [SY05a] and liquid [SY05b] cases. Taking a dense sequence of control meshes as an input, Raveendran et al. [RTWT12] use the velocity of these meshes as boundary conditions on the pressure resolution. Stomakhin and Selle [SS17] use an input control shape in addition to a material flow field in order to enforce boundary conditions and to process particle reseeding around the shape describing an open boundary. In order to generate a character made from fluid, Wiebe and Houston [WH04] present the liquid skin technique that applies a fluid layer over an animated character.

A few methods provide control over fluid simulation using control particles. Rasmussen et al. [REN*04] propose to define soft or hard constraints on several physical properties of fluids using control particles generated by artists. Thürey et al. [TKPR06] use control particles to generate attraction- and velocity-based forces. These forces are thereafter applied to the low-frequency component of the velocity field. A similar strategy has also been applied to smoke [MM13]. After automatically generating control particles by sampling a target shape, Zhang et al. [ZYWL15] provide a position-based control technique to make a controlled fluid match the target. These methods are usually efficient because the control applied on a fluid particle only depends on its control particle neighborhood.

A few intuitive fluid control methods have been proposed. Unlike all the previous work discussed above, the sculpting system presented by Manteaux et al. [MVW*16] enables artists to edit a mesh extracted from a liquid simulation instead of modifying the simulation itself. Their method is able to extract and paste regions of interest on liquid simulation surfaces. Although their sculpting metaphor is shown to be intuitive, their method suffers from a lack of plausibility of the results in many cases. As Manteaux et al. themselves note, physical consistency is not checked and issues may appear when pasting a surface with multiple connected components (e.g., in a splash). The sculpting-inspired tool proposed by Stuyck and Dutré [SD16] also stands out of previous work. Several interesting grid-based methods have been proposed, but they are limited to smoke editing [SDY*15,PM17a] and synthesis [SDN18]. By providing a generation method based on a library of breaking waves, Mihalef et al. [MMS04] describe an intuitive way to generate wave simulations. However, their system only proposes wave profiles in the library and does not allow users to compose different types of liquid animations. Wrenninge and Roble [WR03] introduce a fluid simulation package using the concept of *Water Recorder*. Bojsen-Hansen and Wojtan [BHW16] insert template liquid animations into an existing fluid simulation using special boundary conditions to make the transition smooth. The method is particularly suited to locally edit a region of a precomputed liquid without affecting the rest of the surface. However, newly generated waves and splashes do not propagate outside of the controlled region, which can be an issue for a complex liquid simulation with many parts interacting together.

Note that several techniques have also been proposed to guide a high-resolution simulation using a low-resolution guide. Bergou et al. [BMWG07] simulate a thin shell to make it follow an input low-resolution guide. Similarly, methods have been proposed

to simulate a high-resolution smoke [NCZ*09, NC10] or liquid [NB11, NSG*17] by tracking a low-resolution guide.

In this paper, we introduce an intuitive and easy-to-use liquid control system based on precomputed liquid simulation templates.

3. Our Approach

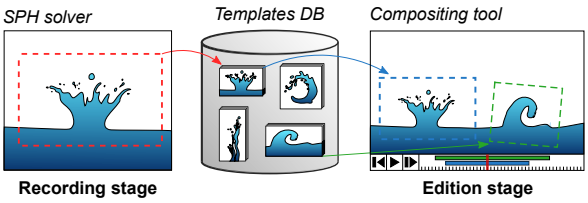

Figure 2: *Overview of our system. Particle-based animation templates are precomputed and stored in a database. The user selects a template and instantiate it with geometrical and temporal transformations. Our system uses all instances of templates to generate control forces in order to drive the current liquid to reproduce a targeted simulation.*

We introduce a method to enable artists to control particle-based liquid simulations using precomputed portions of simulations called *templates*, stored in a database. In our system, a scene consists of SPH particles discretizing either a single-phase or a multi-phase liquid, interacting with rigid bodies. When designing a simulation, a user can pick one or several precomputed animations in the database and edit them using basic affine transformations such as translations, rotations, scalings, or reflections before instantiating them in the scene at given times. Animators can also perform temporal editing over each instance by cropping the beginning and/or the end of each instance to select a temporal region of interest, also changing its playing speed as well as its temporal direction (e.g., forward or backward). Animators can script the motion of each instance in the scene by defining different transformations at keyframes, the transformation affecting each instance at each time step being interpolated between keyframes. When editing a multi-phase liquid simulation, users must specify for each instance the phase to control. Once done, our system generates control forces and manages temporary particles to correctly reproduce the templates in both space and time. Figure 2 shows a schematic overview of our system, while the following sections provide details of its underlying concepts.

3.1. Smoothed Particle Hydrodynamics

In our system, a liquid is simulated using the Smoothed Particle Hydrodynamics method (SPH), which interpolates fluid quantities and spatial derivatives at arbitrary positions with a finite number of sample positions. A quantity A at position $\mathbf{x}$ is approximated using a finite set of known quantities A_i at the positions of its neighboring particles $\mathbf{x}_i$, according to

$$A(\mathbf{x}) = \sum_i \frac{m_i}{\rho_i} A_i W\left(d(\mathbf{x},\mathbf{x}_i), h\right), \tag{1}$$

with m_i and ρ_i respectively the mass and density of the i-th neighboring particle. W is a normalized kernel function that depends on the distance $d(\mathbf{x},\mathbf{x}_i) = \|\mathbf{x}-\mathbf{x}_i\|$ and on the interaction radius h. A kernel function close to Gaussian [Mon92] with a compact support is typically used for computational efficiency. For further information about SPH methods, we refer the reader to the comprehensive state-of-the-art report from Ihmsen et al. [IOS*14].

3.2. Liquid Simulation Template

Our liquid control system uses precomputed templates of liquid animations to generate control over the simulation. Each template corresponds to a liquid simulation of a given duration within a region of space; it is stored in a database. Each template stores the position, velocity, and mass of every particle in the recorded region of space at every frame for the duration of the animation. Global information such as the region size, the animation duration, the template header (e.g., name) are also stored. Templates can come from either realistic or non-realistic liquid animations. Note that no information about solids is stored. Although we also use our system to record templates, any kind of particle-based liquid simulator could be used to record templates, as the information currently stored is standard.

Once recorded and stored as a binary file in the database, each template can be selected, loaded, and placed (i.e., instantiated) in space and time in a scene during the editing process. In order to expand artistic control over the animation, our system allows animators to apply spatial and temporal transformations to the instantiated templates. Once applied in scenes, templates are used to generate control in the areas covered by at least one template, called *controlled areas*. When a template is instantiated, its stored particles become control particles. These particles do not appear as liquid particles in the scene, but instead act as generators of control forces. Therefore, at each frame, the current distribution and properties of control particles, as well as the instance transformations, allow us to exert additional forces that are applied to the scene liquid particles located in every controlled area. Moreover, because not enough liquid particles may be present to reproduce motions in some regions, we also introduce temporary liquid particles to efficiently minimize density differences between template and controlled areas.

In the rest of this paper, simulated liquid particles (including temporary particles) are simply called *liquid particles*. Particles stored in an instantiated template are called *control particles*.

3.3. Control Forces

Each liquid particle located in a controlled area is subject to extra forces, called *control forces*, in order to drive the simulated liquid to locally reproduce the templated liquid behavior.

Attraction and Velocity Forces. In our system, each control particle generates in its neighborhood both an attraction force and a velocity force. These forces are inspired by the work of Thürey et al. [TKPR06]. Attraction forces aim to drive liquid particles into regions covered by control particles. Summing up attraction forces applied on a liquid particle located at position $\mathbf{x}$ yields

$$f_a(\mathbf{x}) = w_a \sum_j \alpha_j \frac{\mathbf{x}_j - \mathbf{x}}{d(\mathbf{x},\mathbf{x}_j)} W\left(d(\mathbf{x},\mathbf{x}_j), h\right), \tag{2}$$

with w_a a constant that defines the strength of the attraction force, $\mathbf{x}_j$ the position of the j-th control particle, and $d(\mathbf{x},\mathbf{x}_j) = \|\mathbf{x}-\mathbf{x}_j\|$. The scaling factor α_j of the j-th control particle reduces the attraction force when control particle j is "covered" by enough liquid particles. The scaling factor is defined as

$$\alpha_j = 1 - \min\left(\sum_i \frac{m_i}{\rho_i} W\left(d(\mathbf{x}_j,\mathbf{x}_i),h\right), 1\right), \tag{3}$$

with mass m_i, density ρ_i, and position $\mathbf{x}_i$ of the i-th neighboring liquid particle. As described by Thürey et al., W is the *Poly6* kernel used in Müller et al. [MCG03].

Since we want the particles to locally exhibit the same kinematics as neighboring control particles, we apply a velocity force to reduce velocity differences between liquid particles and surrounding control particles. The velocity force applied on a particle located at position $\mathbf{x}$ moving with velocity $\mathbf{v}$ is given by

$$f_v(\mathbf{x},\mathbf{v}) = w_v \sum_j (1-\alpha_j)(\mathbf{v}_j - \mathbf{v}) W\left(d(\mathbf{x},\mathbf{x}_j),h\right), \tag{4}$$

with constant w_v defining the strength of the velocity force, and $\mathbf{v}_j$ the velocity of the j-th control particle. Note that, contrary to the original work from Thürey et al. [TKPR06], we multiply the influence of each control particle by its coverage when computing the velocity force. In practice, this coverage factor reduces the impact of velocity forces at the liquid interface, and lets attraction forces bring in more easily some liquid in regions covered by control particles. Both these complementary forces generated from control particles are depicted in Figure 3.

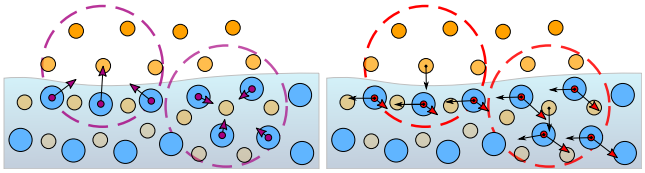

Figure 3: *Left: Attraction forces (purple arrows) generated for two different control particles, and applied on different liquid particles (in blue). Control particles are represented using different shades of orange: light orange for α close to 0, saturated orange for α close to 1. Attraction forces are maximal for control particles with little liquid coverage, and minimal for particles with a significant degree of coverage. Right: Velocity forces (red arrows) generated for the same control particles. Velocities of both control and liquid particles are represented with black arrows. Velocity forces are scaled down when emitted from a control particle with little coverage, since they could counteract the attraction forces.*

Repulsion Layer. Control particles generate both attraction and velocity forces in their neighborhoods in order to attract and drive the controlled liquid to match the behavior of the template. However, we can observe surface dissimilarities between template and simulated liquid in many cases. Indeed, liquid particles that are not covered by control particles are unaffected by them, which means that they can move freely. This lack of control at the template surface causes difficulties to correctly reproduce that surface. To circumvent this limitation, we introduce a repulsion particle layer model illustrated in Figure 4.

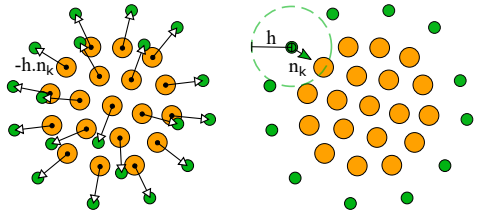

Figure 4: *Repulsion particle layer providing control at the template liquid surface. Left: Seeding repulsion particle candidates (green) using each control particle (orange). Repulsion particle candidates are generated at a distance h along the normalized negative density gradient. Right: Repulsion particles too close to a control particle are discarded. The remaining particles are stored and used to generate repulsion forces collinear to $\mathbf{n}_k$ in their neighborhood.*

This repulsion layer is generated for the whole template duration during a precomputation step that happens when the template is loaded. For each frame of the template and for each control particle, we estimate the control particle density gradient. For each control particle we then generate a repulsion particle at a distance h along the negative density gradient. Repulsion particles too close to a control particle are discarded. In practice, the distance threshold used is set to $a \cdot h$, with $a = 0.9$. Note that each repulsion particle k is oriented, defining its orientation $\mathbf{n}_k$ as the normalized control particle density gradient used to generate it.

Each of these repulsion particles generates a force in its neighborhood, pushing liquid particles towards the template surface, in the direction given by its orientation. Summing up the repulsion forces exerted by repulsion particles on a liquid particle located at position $\mathbf{x}$ close to the template surface yields

$$f_r(\mathbf{x}) = w_r \sum_k \mathbf{n}_k W\left(d(\mathbf{x},\mathbf{x}_k),h\right), \tag{5}$$

with w_r a constant scaling the strength of the repulsion force, $\mathbf{x}_k$ the position of k-th repulsion particle, $\mathbf{n}_k$ its orientation, and W the same kernel used for both attraction and velocity forces. Note that we could save a small portion of precomputation time by generating a repulsion particle only when the l_2-norm of the control particle density gradient is larger than a given threshold. In practice, this threshold is hard to find, leading to either too many particles created or not enough. We observed a smoother and denser repulsion particle layer when using the method described above.

3.4. Temporary Particles

The control forces defined earlier allow us to efficiently reproduce many templates when carefully placed in the scene by an animator. However in some cases, the kinematics of the control particles prevent the liquid to spread into regions covered by liquid in the template. In other cases, the amount of nearby liquid may not be sufficient to rapidly reproduce the whole template.

In order to be able to efficiently and robustly reproduce liquid templates in most cases, without considering whether it is even physically possible, we introduce *temporary particles*. A temporary particle is a liquid particle created by duplicating an existing liquid particle. Such a particle is removed when no longer useful. During its lifetime, the dynamics of a temporary particle is ruled by the Navier-Stokes equations and affected by our control forces.

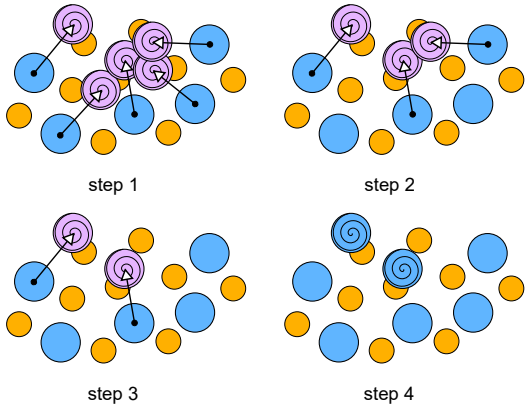

Figure 5: *Seeding temporary particles within a controlled area. Step 1: Each liquid particle (blue) generates a temporary particle candidate (purple, with a spiral pattern) if the density discrepancy between template (orange) and simulation is larger than a specified threshold. Step 2: Candidates too close to a liquid particle are discarded. Step 3: Candidates too close to another candidate are discarded. Step 4: The remaining candidates are transformed into temporary particles (blue, with a spiral pattern).*

At every frame and for every liquid particle in a controlled area, we compute the liquid particle density gradient $\nabla\rho$. For a liquid particle i located at position $\mathbf{x}_i$ and moving at velocity $\mathbf{v}_i$, we trace a position $\mathbf{x}_{tp} = \mathbf{x}_i - d_{tp}\frac{\nabla\rho}{\|\nabla\rho\|}$, with d_{tp} the distance between the liquid particle and the generated candidate, given by $d_{tp} = l \cdot h$. To generate our results, we used $l = 0.5$. We then estimate $\rho(\mathbf{x}_{tp})$ and $\rho_{cp}(\mathbf{x}_{tp})$, respectively the liquid and control particle densities. If the signed difference $\rho_{cp}(\mathbf{x}_{tp}) - \rho(\mathbf{x}_{tp})$ is larger than a given threshold, a temporary particle candidate is generated. During a cleaning pass, candidates are discarded if they are either too close to a liquid particle or to another candidate. Each valid candidate is used to generate a temporary particle with the same physical properties than the particle it came from. Note that temporary particles can also emit themselves temporary particles if needed. In practice, we also disable the generation of temporary particle candidates too close to a solid. The seeding of temporary particles is illustrated in Figure 5.

Because we globally and ultimately want to respect the law of conservation of mass, we need to remove temporary particles when they become unnecessary. This happens when a temporary particle leaves every controlled area. However, this cannot be too sudden as it may lead to visual artifacts (e.g., a splash consisting mainly of temporary particles could disappear in mid-air as soon as it leaves a controlled area). In practice, we remove a temporary particle as soon as the density of its surrounding original (non-temporary) liquid particles is higher than a given threshold. The higher this threshold, the longer temporary particles will continue to interact with fluid particles outside every controlled area. Similarly to the creation process, we also enforce a minimal distance between candidates to deletion.

In practice, temporary particles extend the spectrum of possibilities from an artistic standpoint, and proved to be a simple, flexible, and efficient way to simulate physically inspired animations. Seeding and removing temporary particles respectively follow Algorithms 1 and 2.

Algorithm 1: Seeding temporary particles

```
Candidates = ∅;
for i ∈ Particles do
    emitted = FALSE;
    for s ∈ Instances do
        if x_i inside s then
            ∇ρ ← computeDensityGrad(x_i);
            x_tp ← x_i − d_tp ∇ρ/‖∇ρ‖;
            x_np ← getNearestParticlePos(x_tp);
            if ‖x_tp − x_np‖ < d_tp − ε then
                break;
            ρ ← computeDensity(x_tp);
            ρ_cp ← computeCpDensity(x_tp);
            if ρ_cp − ρ > η then
                Candidates = Candidates ∪ {(x_tp, v_i)};
                emitted = TRUE;
                break;
        if emitted then
            break;
for c ∈ Candidates do
    x_nc ← getNearestCandidatePos(c);
    if ‖x_np − x_c‖ < d_tp then
        Candidates = Candidates \ c;
for c ∈ Candidates do
    p ← generateParticle(x_c, v_c);
    Particles = Particles ∪ p;
    TemporaryParticles = TemporaryParticles ∪ p;
```

Algorithm 2: Removing temporary particles

```
Candidates = ∅;
for tp ∈ TemporaryParticles do
    if outsideEveryInstances(tp) then
        ρ ← computeNonTemporaryDensity(x_tp);
        x_nc ← getNearestCandidatePos(x_tp);
        if ρ > τ ∧ ‖x_nc − x_tp‖ > d_tp then
            Candidates = Candidates ∪ tp;
for c ∈ Candidates do
    removeParticle(c);
```

3.5. Animation-editing Metaphors

Affine Transformation. In order to propose a comprehensive tool, we allow for time-varying affine transformations of our controlled areas. As mentioned earlier, standard translations, rotations, both uniform and nonuniform scalings, and reflections can thus be used and combined to modify template instances. In our system, each template instance contains a reference to the original template, as well as the affine transformation applied to it. Every time we want to evaluate a physical quantity of a transformed template instance at a given position and time, we apply the inverse transformation of the instance to this position in order to estimate this quantity into the untransformed (canonical) space. Control particle density is thus evaluated in the untransformed space. Similarly, attraction and repulsion forces are evaluated in the untransformed space, before applying the instance transformation to get the final forces. Velocity forces are computed the same way after transforming each liquid particle velocity using the inverse of each instance transformation. Estimation of density and forces is illustrated in Figure 6. In the case of an instance modified using a time-varying affine transformation, each control particle velocity is also virtually modified by summing the velocity of the instance at its world-space position.

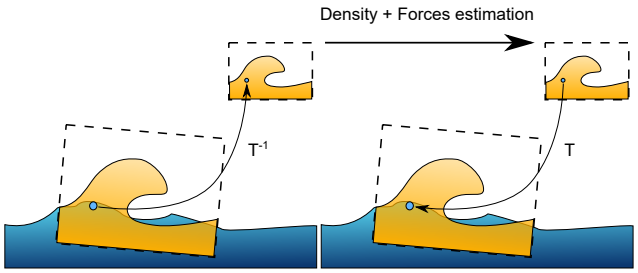

Figure 6: *Estimation of density and forces in a transformed template instance. The template instance is transformed by* $\mathbf{T}$ *in the scene. A position in the scene space is transformed by the inverse transformation* $\mathbf{T}^{-1}$ *in template canonical space to compute density of control particles as well as control forces.*

Control Falloff. To reduce artifacts due to control discontinuities at the borders of a controlled area, we extend a falloff area around it. In this falloff area, temporary particles cannot be created nor removed. We also decrease control forces by replacing w_κ by $w_\kappa(t) = w_\kappa t^\gamma$, with $t \in [0,1]$ a linear coefficient defining the position along the cross section of the falloff area, $\kappa \in \{a,v,r\}$ the force type (respectively, attraction, velocity, and repulsion), and γ the decay parameter. Both values are user-defined parameters that affect how control is attenuated as soon as liquid particles leave the interior region of control areas.

Temporal Control. In our system, each template instance starts and ends affecting the simulated liquid at moments chosen by the user. In order to allow users to modify these temporal boundaries, we provide an animation-editing-like user interface. In our interface, each template instance is represented as a rectangle, similar to a track in an animation-editing software. Each instance can be temporally edited by translating or cropping the activation period of a template instance. In addition, a user can modify the play speed of a template instance, running the animation at half the speed of the original template for example. In such a case, we virtually modify each control particle velocity by multiplying it by the play speed factor. Finally and if a template instance is played in reverse order (i.e., from the end to the beginning), velocities of control particles are simply multiplied by -1 when evaluated. We believe that this familiar temporal animation-editing metaphor leads to simplicity and ease of use in global animation management.

Simulation and Template Frequencies. A template is captured at its animation speed with its own density of particles. The data is stored at every frame. When applied in a scene, a template can be scaled in space and time. Working in the original canonical space of a template allows for smoother transitions and correct physical quantities estimation. However, a user must be aware that in particular, temporal frequencies of a template cannot be scaled up or down too strongly (for example by a factor of 100 or more, or 0.01 or less) when applying it in a scene. During simulation, a frame of simulation will be computed as a certain number of time steps. A template generates its forces only once per frame, and its forces are applied at every simulation time step.

4. Implementation

Our methods for recording and compositing liquid simulations have been implemented in a system based on the open-source library SPlisHSPlasH [Ben16]. Although we tested our liquid simulations on a number of SPH methods, we used DFSPH [BK15] to generate the results illustrated in this paper and in the accompanying video. This choice is motivated by its stability as well as the small number of solver iterations required during simulation. We also used the two-way coupling method of Akinci et al. [AIA*12]. Our graphical user interface, shown in Figure 7, is built on ImGui [Cor20].

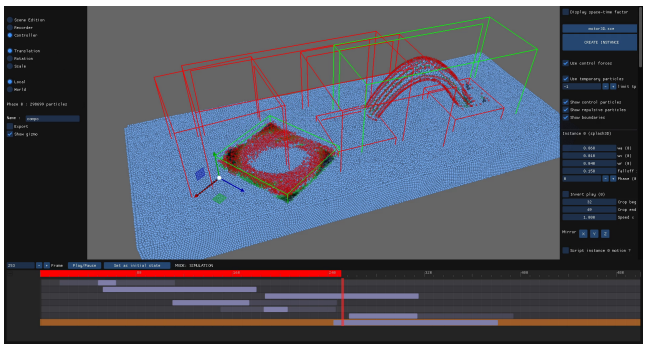

Figure 7: *GUI of our system.*

Once recorded, each template is stored as a binary file. In our compositing tool, templates are always loaded once in memory, even when instantiated several times in the scene. During simulation and in order to speed up particle search, we use several uniform acceleration structures usually called buckets in the literature. Two buckets covering the whole simulation domain are used to store liquid particles and temporary particle candidates. In addition to those buckets, two local buckets covering each template are used to speed up control and repulsion particle search. When loading a template, control and repulsion particles are sorted in those buckets

for the whole template duration, allowing efficient particle neighbor searching at runtime.

During the scene (i.e., liquid and rigid bodies) simulation, the state of each element of the scene is stored at each frame in memory, in order to allow fast replay of an art-directed animation. This is particularly useful to visualize and appreciate the complex motions of liquids, after slower editing in large 3D domains. Our system is a multi-threaded CPU implementation using OpenMP.

To design our 3D examples, we initialized the weights of our forces using the following set of default values: $w_a = 0.06$, $w_v = 0.01$, $w_r = 0.04$. We then slightly refined those parameters during a short iterative process. In 2D, the initial set of parameter values used is $w_a = 0.2$, $w_v = 0.03$, $w_r = 0.1$. For all the examples, we used $\eta = \tau = 0.35\rho_0$ as density thresholds, with ρ_0 the rest density.

5. Results

5.1. 2D Proof of Concept

We used our system to generate many controlled liquid animations. As a proof of concept and in order to illustrate a number of aspects of our system, we first tested it on 2D examples. These 2D examples are simulated mostly at interactive framerates, ranging from 5.3 to 30.8 frames per second (see column *Simulation* in Table 1). Note that we use DFSPH [BK15] in SPlisHSPlasH [Ben16] with normal settings for simulation steps, without lowering its simulation quality, even in our editing mode.

We show images of particles and bounding boxes in 2D in order to better display various features of liquid and control particles, templates, etc. All images in the following figures come from our system and are laid out in the same configuration. On the left, the simulation domain with the template overlaid, its bounding box and surrounding falloff box. Simulation particles are shown in light blue, control particles in a color gradient ranging from green (for α close to 0 in Equation (3)) to black (for α close to 1), and repulsion particles in red. On the right, all simulation particles, without the control particles. On the middle slightly above, the template as recorded in its own simulation, and the bounding box of its domain (in red).

In a first simple example depicted in Figure 8, we recreate the impact of a rectangular drop of water falling in a pool. We wanted to show the impact of a drop of water, although without inserting new particles that would need to appear in mid-air. Our method correctly modifies the liquid behavior of the simulation to locally recreate the drop impact and propagating ripples, without visual artifacts due to the drop during its fall.

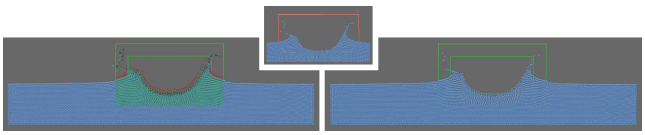

Figure 8: *2D Rectangular drop.*

We recorded a wave template from a wave-machine-based simulation. Once instantiated in the scene, the template is correctly reproduced as shown in Figure 9, giving birth to a wave that propagates out of the controlled region in a realistic way. Thanks to the falloff area, boundaries are smooth between controlled and uncontrolled liquids, leading to a convincing blended result.

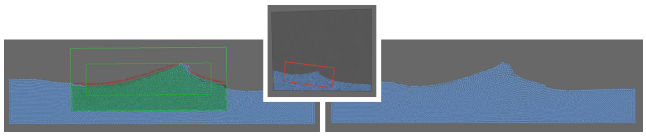

Figure 9: *2D Wave machine.*

Our method is compatible with templates containing solid objects. Thanks to its layer of repulsion particles, our method provides stronger control in boundary-liquid template regions in order to correctly reproduce these regions. We demonstrate this aspect in a simple 2D example illustrated in Figure 10. We instantiate a template storing the behavior of a liquid mixed by a rotating blade. Not using repulsion particles (top image on the right) shows a smoother relative displacement of liquid particles, while using them (bottom image on the right) follows more closely the blade boundaries.

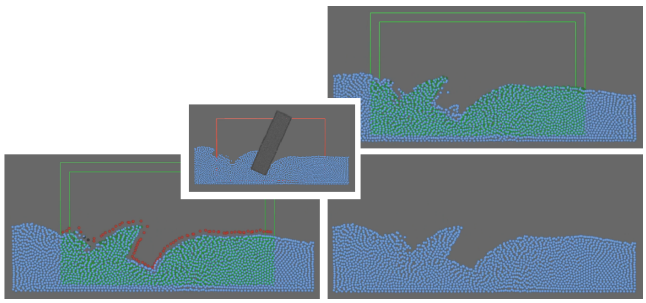

Figure 10: *2D Rotating blade.*

We recorded a 2D template in which a thin water jet, unaffected by gravity, evolves through the simulation domain while hitting disks as obstacles. Although this scenario shown in Figure 11 is unrealistic, we can instantiate it in a scene, flip it upside-down, and our system faithfully reproduces the behavior of the template, even under gravity. Note how the repulsion particles (in red) help to channel and constrain the liquid particles where the jet has a sharper surface.

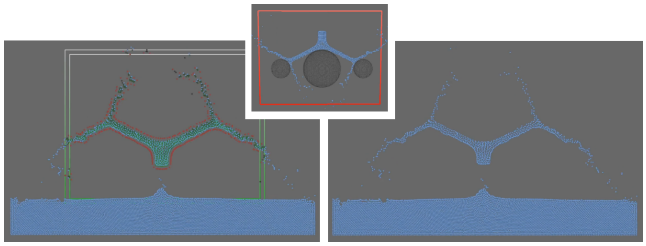

Figure 11: *2D Zero gravity.*

Multiple examples demonstrating how a user can manipulate a

template using both isometric and non-isometric geometric transformations, as well as temporal editing, can be found in the accompanying video. Allowing a user to easily manipulate templates extend the spectrum of possibilities from an artistic standpoint.

5.2. Going to 3D

Our 3D examples are simulated in less interactive framerates, due to much larger sets of liquid particles. It goes from 0.8 to 48.5 seconds per frame (see column *Simulation* in Table 1). This is where an extension to GPU support would come handy (see Section 6).

3D Rotating blade. Several 3D simulations have been generated using our method. Figure 12 shows a scene where a calm surface of water is perturbed with a template describing the dynamics of a liquid mixed by a blade rotating horizontally. Similarly to our 2D experiment, our system efficiently and correctly reproduces the template in both space (region covered by the template instance) and time (over the duration of the template). Liquid particles spin away from the center, creating a hole and partly-circular waves.

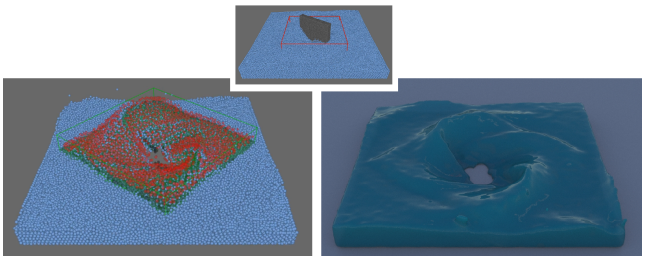

Figure 12: *3D Rotating blade.*

Viscous. Our method is compatible with templates simulated under physical properties different than the controlled liquid. In Figure 13, we control a water-like liquid to follow a buckling behavior recorded from a viscous honey-like liquid. The template is scaled by a factor of two in all three dimensions and flipped upside-down in order to make a buckling pattern magically rise from the water surface. Although viscous liquids and water exhibit very different behaviors, we can still control a water simulation, creating the buckling pattern in mid-air.

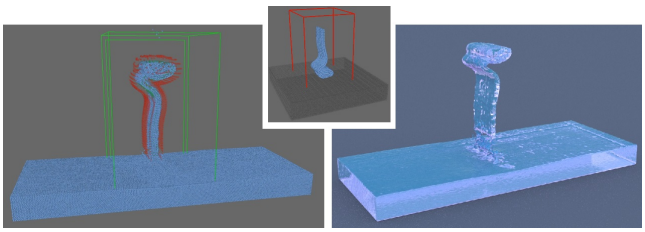

Figure 13: *3D Viscous.*

Letters. When loaded in memory, templates are played in their usual time-forward direction by default. However, users can play a template in reverse order. This feature is demonstrated in a 3D example where we recreate the shapes of original meshes with a controlled liquid. During a recording phase, water-filled letter shapes fall and splash at the bottom of the simulation domain. In those simulations (one for each letter), we slightly reduced gravity in order to lead to smaller splashes. After recording the simulations, we instantiated the templates in a scene, playing them from the end (liquid at the bottom of the simulation domain) to the beginning (liquid forming letters). Our method efficiently reproduces the animation, drawing water from the overlapping simulation domain to recreate the letters. A frame from this scene is illustrated in Figure 1.

Multiphase. Our system is compatible with the control of multiphase liquids. When a template is instantiated in the scene, a user can select which liquid phase the template instance controls. Figure 14 shows an art-directed simulation generated using our method. Two instances are generated in the scene (one being scaled along the Y axis), each of them controlling a different phase of a two-phase liquid. The two liquids are pulled out of their respective spaces, jump in mid-air, and dive back in the liquids, partially mixing at the bottom of the reservoir with an expected behavior.

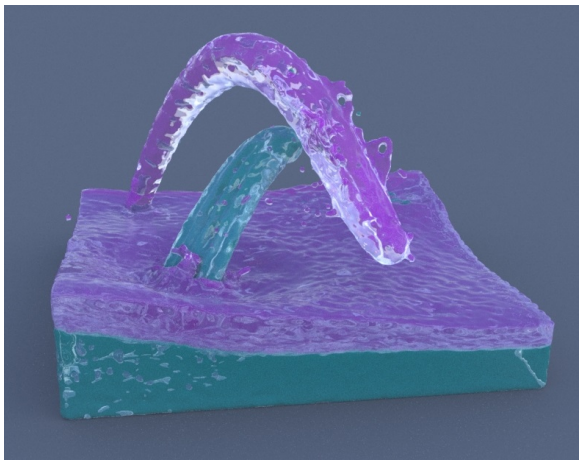

Figure 14: *3D Multiphase.*

3D Boat. In our system, a controlled liquid can interact with rigid bodies. In Figure 15, a hand made of water raises a floating boat in mid-air. With a strategy similar to the one from Figure 1, a water-filled hand shape has been recorded, falling and splashing in an empty scene. This template is played backward while controlling liquid, in order to make a hand emerge from underneath. In this simulation with around 2.49M particles, the water from the hand interacts with the boat using classical two-way coupling.

5.3. Statistics

Table 1 gives statistics about the simulations presented in this paper and in the accompanying video. All the simulations were run on a desktop PC with an i9-9900K 8-core CPU at 3.60 GHz with 128 GB of memory.

The numbers of control, repulsion, and temporary particles are given as an average per frame, for all frames with at least one template activated. These particles are not included in the number of particles in the *Liquid* column, the latter being constant for each

Scene	Fig.	Particles (avg number per frame) Liquid	Control	Repulsion	Temporary	Timings (ms per frame) Simulation	Control	Memory (GB)	Storage (MB)
2D Rectangular drop	8	6.7k	1.3k	88	36	190.2	5.6 (2.9%)	0.33	3.87
2D Wave machine	9	6.7k	2.3k	130	14	128.7	13.7 (10.6%)	0.34	7.44
2D Rotating blade	10	2.2k	1.3k	79	0	32.4	6.7 (20.7%)	0.24	12.0
2D Water jet	—	6.7k	1.3k	217	138	96.7	3.9 (4.0%)	0.35	4.19
2D Zero gravity	11	6.7k	0.6k	209	0	116.5	3.4 (2.9%)	0.46	3.02
2D Double dam	—	6.7k	1.0k	92	43	143.1	4.0 (2.8%)	0.46	3.50
3D Viscous	13	298.7k	1.3k	713	76	6566.5	53.2 (0.8%)	5.81	5.11
3D Rotating blade	12	56.2k	20.0k	2.3k	0	807.2	135.4 (16.8%)	2.4	82.7
3D Letters	1	298.7k	39.8k	8.2k	329	5207.5	474.8 (9.1%)	7.4	274.1
3D Multiphase	14	106.4k	2.6k	1.4k	74	2274.5	125.7 (5.5%)	3.2	5.12
3D Boat	15	2.49M	363.3k	30.1k	994	44585.2	3921.6 (8.8%)	80.6	3488.0
3D Composition	—	298.7k	12.2k	1.7k	43	4900.1	112.4 (2.3%)	7.8	90.0

Table 1: *Various statistics on our controlled simulations. Some simulations without a figure are only part of the accompanying video.*

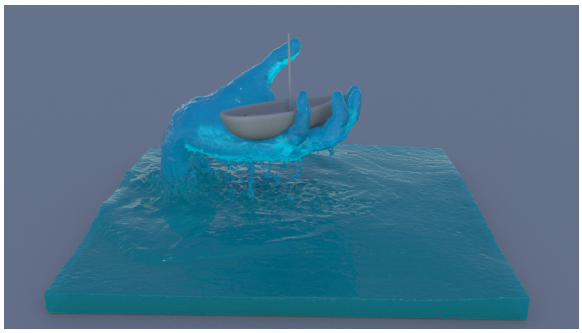

Figure 15: *3D Boat.*

simulation. The number of control particles is the average of particles stored in its template. This number, multiplied by the number of frames for the duration of its template, gives the relative size of the file stored on disk, shown in column *Storage*. The ratios Control/Liquid particles vary from 9.0% to 59.1% in 2D, and 0.43% to 35.6% in 3D. The number of temporary particles per scene is relatively low, but proved important in special cases with high discrepancy between the coverages template-scene. The number of repulsion particles depends on the proportion of surfaces in the template.

Timings are given as an average per frame during an activated template. Control is applied once per frame, while simulation with the DFSPH [BK15] method requires an adaptive number of steps. Control in 2D adds from 2.8% to 20.7% to the simulation time, and in 3D from 0.8% to 16.8%.

The *Memory* column includes all particles, templates, acceleration structures, recorded simulation to replay in real time, and the system itself, with its libraries. The system without any liquid nor template occupies 71.2 MB.

6. Conclusions and Future Work

Liquid animations are notoriously difficult to design. Indeed, a fully physical simulation offers too little control by setting initial configurations and physical properties, while a fully artist-based animation may not look as realistic. We described our intuitive tool that allows artists to compose SPH-based liquid simulations using precomputed liquid simulation templates. Our system generates forces from control particles issued from animation templates in order to drive a global liquid simulation. When desired, temporary particles are automatically and seamlessly added and removed when insufficient liquid can fill empty regions. Our system is easy to use, robust, efficient, and predictable. To demonstrate its potential, we created and analyzed several animations that would have been very complex to generate without our method.

While we are generally happy with our prototype system, there are several improvements to make it even more efficient and robust.

We used basic acceleration structures and multi-threading on CPU to reach interactive results on simple simulations. Our system adds little overhead on top of our SPH simulation. We have not yet looked at going from CPU-only to GPU, but it should deliver reasonable gains since our control strategy can be easily and efficiently parallelized.

While breaking physical correctness, the notion of temporary particles has provided flexibility to follow more closely some template simulations. However, because we rely on local considerations to create and remove temporary particles, we are not globally optimal. Using global considerations or optimizing over a wider time window would lead to more optimal solutions, including fewer temporary particles, but at an increased computational cost.

Our system is not designed to handle transformations involving large scaling factors, and side effects can appear. Our method should be improved to robustly handle these extreme deformations.

The placement of template instances is sometimes a little challenging in order for them to be seamlessly integrated in a liquid. Automatic position and refinement of control parameters over time could be investigated to improve this aspect.

More complex ways to combine templates could also be found, allowing a user to subtract templates or attach them to some fluid features. Data compression could also be relevant to efficiently store and load templates involving hundreds of thousands particles.

Other directions for future investigations include editing other types of fluids (e.g., smoke or fire), and applying our method to multiresolution flows. Since we are highly interested in art-directed control, performing a user-study with artists specialized in fluid animation would be relevant.

Acknowledgements

Financial support was provided by NSERC and our respective universities.

References

[AIA*12] AKINCI N., IHMSEN M., AKINCI G., SOLENTHALER B., TESCHNER M.: Versatile rigid-fluid coupling for incompressible SPH. *ACM Trans. Graph. 31*, 4 (2012), 62. 6

[Ben16] BENDER J.: SPlisHSPlasH, 2016. URL: https://github.com/InteractiveComputerGraphics/SPlisHSPlasH. 6, 7

[BHW16] BOJSEN-HANSEN M., WOJTAN C.: Generalized non-reflecting boundaries for fluid re-simulation. *ACM Trans. Graph. 35*, 4 (2016). 2

[BK15] BENDER J., KOSCHIER D.: Divergence-free smoothed particle hydrodynamics. In *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation* (2015), p. 147–155. 6, 7, 9

[BMWG07] BERGOU M., MATHUR S., WARDETZKY M., GRINSPUN E.: TRACKS: Toward directable thin shells. *ACM Trans. Graph. 26*, 3 (2007), 50. 2

[Cor20] CORNUT O.: Dear ImGui: Bloat-free immediate mode graphical user interface for C++ with minimal dependencies, 2020. URL: https://github.com/ocornut/imgui. 6

[FL04] FATTAL R., LISCHINSKI D.: Target-driven smoke animation. *ACM Trans. Graph. 23*, 3 (2004), 441–448. 2

[FM97] FOSTER N., METAXAS D.: Controlling fluid animation. In *Proc. Comput. Graph. International* (1997), IEEE, pp. 178–188. 2

[HK04] HONG J.-M., KIM C.-H.: Controlling fluid animation with geometric potential. *Comput. Animat. Virtual Worlds 15*, 3-4 (2004), 147–157. 2

[IEGT17] INGLIS T., ECKERT M.-L., GREGSON J., THUEREY N.: Primal-dual optimization for fluids. *Comput. Graph. Forum 36*, 8 (2017), 354–368. 2

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: SPH fluids in computer graphics. In *Eurographics - State of the Art Reports* (2014). 3

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation* (2003), pp. 154–159. 4

[MM13] MADILL J., MOULD D.: Target particle control of smoke simulation. In *Proc. Graphics Interface* (2013), pp. 125–132. 2

[MMS04] MIHALEF V., METAXAS D., SUSSMAN M.: Animation and control of breaking waves. In *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation* (2004), pp. 315–324. 2

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30*, 1 (1992), 543–574. 3

[MTPS04] MCNAMARA A., TREUILLE A., POPOVIĆ Z., STAM J.: Fluid control using the adjoint method. *ACM Trans. Graph. 23*, 3 (2004), 449–456. 2

[MVW*16] MANTEAUX P.-L., VIMONT U., WOJTAN C., ROHMER D., CANI M.-P.: Space-time sculpting of liquid animation. In *Proc. Motion in Games* (2016), ACM, pp. 61–71. 2

[NB11] NIELSEN M. B., BRIDSON R.: Guide shapes for high resolution naturalistic liquid simulation. *ACM Trans. Graph. 30*, 4 (2011), 83. 3

[NC10] NIELSEN M. B., CHRISTENSEN B. B.: Improved variational guiding of smoke animations. *Comput. Graph. Forum 29*, 2 (2010), 705–712. 3

[NCZ*09] NIELSEN M. B., CHRISTENSEN B. B., ZAFAR N. B., ROBLE D., MUSETH K.: Guiding of smoke animations through variational coupling of simulations at different resolutions. In *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation* (2009), p. 217–226. 3

[NSG*17] NIELSEN M. B., STAMATELOS K., GRAHAM A., NORDENSTAM M., BRIDSON R.: Localized guided liquid simulations in Bifrost. In *ACM SIGGRAPH Talks* (2017). 3

[PHT*13] PAN Z., HUANG J., TONG Y., ZHENG C., BAO H.: Interactive localized liquid motion editing. *ACM Trans. Graph. 32*, 6 (2013), 184. 2

[PM17a] PAN Z., MANOCHA D.: Editing smoke animation using a deforming grid. *Computational Visual Media 3*, 4 (2017), 369–378. 2

[PM17b] PAN Z., MANOCHA D.: Efficient solver for spacetime control of smoke. *ACM Trans. Graph. 36*, 5 (2017), 162. 2

[REN*04] RASMUSSEN N., ENRIGHT D., NGUYEN D. Q., MARINO S., SUMNER N., GEIGER W., HOON S., FEDKIW R.: Directable photorealistic liquids. In *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation* (2004), pp. 193–202. 2

[RTWT12] RAVEENDRAN K., THUEREY N., WOJTAN C., TURK G.: Controlling liquids using meshes. In *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation* (2012), pp. 255–264. 2

[RWTT14] RAVEENDRAN K., WOJTAN C., THUEREY N., TURK G.: Blending liquids. *ACM Trans. Graph. 33*, 4 (2014). 2

[SD16] STUYCK T., DUTRÉ P.: Sculpting fluids: A new and intuitive approach to art-directable fluids. In *ACM SIGGRAPH Posters* (2016), p. 11. 2

[SDN18] SATO S., DOBASHI Y., NISHITA T.: Editing fluid animation using flow interpolation. *ACM Trans. Graph. 37*, 5 (2018), 173. 2

[SDY*15] SATO S., DOBASHI Y., YUE Y., IWASAKI K., NISHITA T.: Incompressibility-preserving deformation for fluid flows using vector potentials. *The Visual Computer 31*, 6-8 (2015), 959–965. 2

[SS17] STOMAKHIN A., SELLE A.: Fluxed animated boundary method. *ACM Trans. Graph. 36*, 4 (2017), 68. 2

[SY05a] SHI L., YU Y.: Controllable smoke animation with guiding objects. *ACM Trans. Graph. 24*, 1 (2005), 140–164. 2

[SY05b] SHI L., YU Y.: Taming liquids for rapidly changing targets. In *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation* (2005), pp. 229–236. 2

[Thu16] THUEREY N.: Interpolations of smoke and liquid simulations. *ACM Trans. Graph. 36*, 1 (2016). 2

[TKPR06] THÜREY N., KEISER R., PAULY M., RÜDE U.: Detail-preserving fluid control. In *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation* (2006), pp. 7–12. 2, 3, 4

[TMPS03] TREUILLE A., MCNAMARA A., POPOVIĆ Z., STAM J.: Keyframe control of smoke simulations. *ACM Trans. Graph. 22*, 3 (2003), 716–723. 2

[WH04] WIEBE M., HOUSTON B.: The Tar monster: Creating a character with fluid simulation. In *ACM SIGGRAPH Sketches & Appl.* (2004). 2

[WR03] WRENNINGE M., ROBLE D.: Fluid simulation interaction techniques. In *ACM SIGGRAPH Sketches & Appl.* (2003). 2

[ZYWL15] ZHANG S., YANG X., WU Z., LIU H.: Position-based fluid control. In *Proc. Symp. Interact. 3D Graph. Games* (2015), ACM, pp. 61–68. 2