Physically Guided Liquid Surface Modeling from Videos

Huamin Wang* Miao Liao† Qing Zhang† Ruigang Yang† Greg Turk*

Georgia Institute of Technology* University of Kentucky†

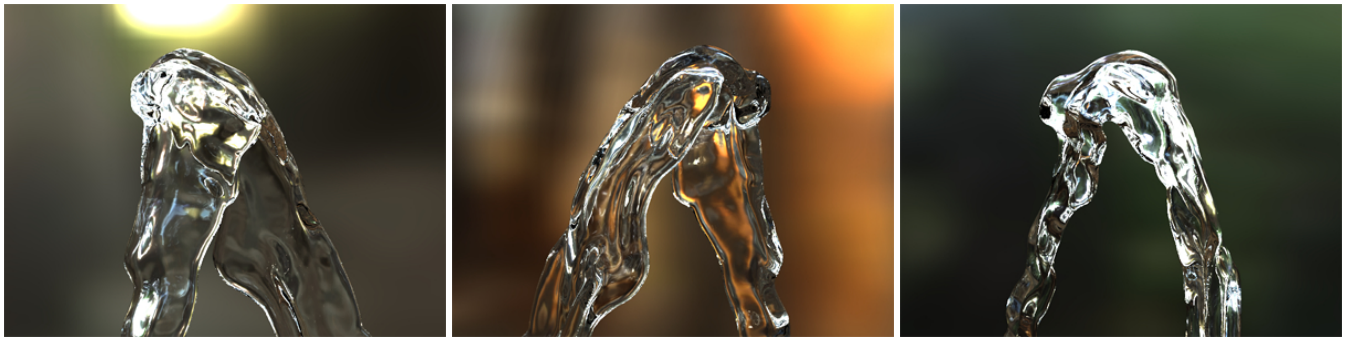

Figure 1: *A synthetic rendering of a 3D model reconstructed from video of a fountain. These are three static views of the same time instant.*

Abstract

We present an image-based reconstruction framework to model real water scenes captured by stereoscopic video. In contrast to many image-based modeling techniques that rely on user interaction to obtain high-quality 3D models, we instead apply automatically calculated physically-based constraints to refine the initial model. The combination of image-based reconstruction with physically-based simulation allows us to model complex and dynamic objects such as fluid. Using a depth map sequence as initial conditions, we use a physically based approach that automatically fills in missing regions, removes outliers, and refines the geometric shape so that the final 3D model is consistent to both the input video data and the laws of physics. Physically-guided modeling also makes interpolation or extrapolation in the space-time domain possible, and even allows the fusion of depth maps that were taken at different times or viewpoints. We demonstrated the effectiveness of our framework with a number of real scenes, all captured using only a single pair of cameras.

CR Categories: I.3.5 [COMPUTER GRAPHICS]: Computational Geometry and Object Modeling—Physically based modeling; I.3.8 [COMPUTER GRAPHICS]: Three- Dimensional Graphics and Realism—Animation

Keywords: image-based reconstruction, space-time model completion, physically-based fluid simulation

1 Introduction

In recent years, modeling complex real world objects and scenes using cameras has been an active research topic in both graphics and vision. Exemplary work in this broad topic includes reconstructing flower models [Quan et al. 2006], tree models [Tan et al. 2007], hairs [Wei et al. 2005], urban buildings [Sinha et al. 2008; Xiao et al. 2008], human motion [Zitnick et al. 2004a; de Aguiar et al. 2008], and cloth [White et al. 2007; Bradley et al. 2008]. The typical approach is to use one or more cameras to capture different views, from which the 3D shape information of the scene can be estimated by matching feature points. User interaction is often required to refine the initial 3D shape to create high-quality models. Missing from the list of objects that have been successfully reconstructed from video is water. Water's complex shape, frequent occlusions, and generally non-Lambertian appearance cause even the best matching methods to yield poor depth maps. Its dynamic nature and complex topological changes over time make human refinement too tedious for most applications.

In computer graphics, a common technique to produce water animation is physically-based fluid simulation, which is based on simulating fluid dynamics from the initial state of a fluid scene. While realistic water animation can be generated by various numerical simulation approaches, these approaches can suffer from numerical errors that accumulate over time, including volume loss and loss of surface details. The computational cost is another issue in physically based fluid simulation, since the governing partial differential equations are expensive to solve and the time steps need to be sufficiently small to maintain stability and accuracy.

In this paper we present the idea of combining physically-based simulation with image-based reconstruction to model dynamic water from video. That is, we adapt physically-based methods as a correction tool to refine the water surface that is initially generated from matching feature points. In order to enforce temporal coherence, we develop a 3D flow estimation method to approximate the velocity flow between two reconstructed shapes in neighboring frames. The surface optimization method then removes redundant errors, applies physically based constraints such as volume preservation and smoothness, and completes the shape sequence by filling in missing fluid regions. In this way, the final dynamic water model matches the fidelity of the real world and the results are physically sound, even though fluid dynamics may not be strictly enforced in certain cases. Since fluid dynamics is only used as a

*e-mail: {whmin, turk}@cc.gatech.edu

†e-mail: {mliao3, qzhan7, ryang}@cs.uky.edu

constraint rather than the target function to derive the entire surface sequence, this process is efficient and should be easy to accelerate using graphics hardware.

Incorporating the physical properties of fluid provides strong constraints on the possible water surface shape. This means the quality and coverage requirement for the initial 3D shape is significantly reduced. This allows us to generate plausible 3D water surface models even when observed by just one stereo camera, that is, when more than 50% of the surface is occluded. A single-depth-view solution is much easier to set up and use than the typical requirement of a surrounding array of cameras.

To help with spatial feature matching, we add white paint to the water to avoid refraction, and we use a light projector to place a fixed random texture pattern onto the moving fluid surfaces. The equipment used for our capturing system is shown in Figure 3. Note that our surface optimization scheme is not tied to any particular depth acquisition method, nor does it require appearance-based feature tracking over time, and this makes the use of active range sensing methods possible. Our choice of using a stereo camera in this system is due to two considerations. First, since this technique can be potentially used with any capturing device, it is interesting to test its performance in a tough case when less surface information is provided. Second, one part of our ultimate goal of this line of research is to reconstruct large, outdoor fluid phenomena, in which case a hand-held stereo camera is much more practical than a surrounding multi-camera system.

We view this new approach for creating fluid models as an alternative to creating fluid animation through direct simulation. As with other camera-based data capture methods, our approach has the benefit of capturing the nuances and details of fluids that may be difficult to achieve using simulation alone. With an explicit 3D model, the captured water can be re-rendered seamlessly with other graphics models. Though not demonstrated in this paper, our framework should allow artists to design and modify a coarse initial shape in order to create stylized animations. Therefore we believe that our method may have applications in feature-film special effects and in video game content creation.

2 Related Work

In graphics, Hawkins et al. [2005] demonstrated how to reconstruct smoke animation using a specialized capture system that included a laser and a high-speed camera. Morris and Kutulaskos [2007] and Hullin et al. [2008] successfully reconstructed static transparent objects such as a vase or a glass by tracing light transport under structured scanning. A time-varying height-field surface can also be reconstructed by searching refractive disparity as proposed by Morris and Kutulakos [2005] when the light is refracted only once. Water thickness can be measured when water is dyed with fluorescent chemical as shown by Ihrke et al. [2005], in which case the liquid surface is calculated as a minimum solution of a photo consistency based error measure using the Euler-Lagrangian formulation. Atcheson et al. [2008] used Schlieren tomography to capture fluids with time-varying refraction index values, such as heated smoke. In general, these techniques require specific capture devices for certain fluid effects and they consider the fluid shape in each frame independently as a static reconstruction problem.

Bhat et al. [2004] studied how to synthesize new fluid videos by tracing textured 2D particles over existing video sequences according to temporal continuity. This is an image editing approach, and no 3D model is created. Given surrounding reliable scanned data without redundant errors, Sharf et al. [2008] successfully used incompressible flow to complete shape animations with spatial-temporal coherence, assuming that the surface moves less than one grid cell in each time step. Although fluid animation also satisfies incompressibility with spatial-temporal coherence, the problem we address in this paper is more challenging since the input is captured from a single stereo camera, so our initial water surface contains outliers and substantially more missing parts than from a surrounding setup. Furthermore the water surface can move significantly more than one grid cell due to the capture rate and grid resolution.

Foster and Metaxas [1996] studied how to generate fluid animation as an application of computational fluid dynamics. Shortly after this, the stable fluid method introduced by Stam [1999] used the semi-Lagrangian method to handle fluid velocity advection. In a series of papers, Enright, Fedkiw and Foster [2001; 2002] used the level set method and particles to evolve liquid surfaces for more complex liquid motions. Besides volumetric representation, water animation can also be simulated using particle systems or tetrahedral meshes. When using fluid dynamics, numerical errors including volume loss and detail loss can be a problem in physically based fluid simulation. Another problem in this area is how to reach stylized fluid shapes at specified time instants. McNamara et al. [2004] and Fattal and Lischinski [2004] studied how to constrain fluid simulation by creating artificial control forces. In our problem, we consider instead how to improve reconstructed models by physically-based constraints.

Researchers in other domains have used various techniques to capture the behavior of fluids. The fluid imaging community regularly makes use of the Particle Imaging Velocimetry (PIV) method to capture flow fields from the real world by tracking macroscopic particles mixed in the fluid. This method creates fluid velocity values in the *interior* of bodies of water, but the approach cannot be used to reconstruct the geometry of the water's *surface* due to the difficulty in maintaining the distribution of those particles in the fluid. More details of the PIV method can be found in [Grant 1997].

To obtain a complete space+time model from dynamic scenes, a camera array system is usually deployed to capture objects from different views (e.g., [Kanade et al. 1999; Simon et al. 2000; Zitnick et al. 2004b]). Recently using a sparse camera array becomes an active research topic (e.g., [Wand et al. 2007; Mitra et al. 2007; Sharf et al. 2008]. We push this trend to the limit by using only one pair of cameras with a narrow baseline. We show that by incorporating physically-based constraints, the amount of input data needed for complete 4D modeling can be dramatically reduced.

3 Overview

Given a video sequence captured by a synchronized, calibrated stereo camera system, the goal of our hybrid water modeling technique is to efficiently reconstruct realistic 3D fluid animations with physical soundness. Our framework consists of two stages as shown in Figure 2. Let $\{I_t\}$ and $\{J_t\}$ be video sequences captured by stereo camera at time $t \in [0, T]$. In the first stage, an initial shape sequence $\Psi = \{\psi_i\}$ is assembled by reconstructing each frame independently using depth from stereo. Ψ is then optimized in the second stage to generate a shape sequence $\Phi = \{\phi_i\}$ that satisfies spatial-temporal coherence.

Spatial coherence means that Φ should be similar to the captured data input Ψ. Temporal coherence means that Φ should satisfy the behavior of fluids as much as possible. Mathematically, this is interpreted as the minimum solution to the following energy functional:

$$\mathbb{E}(\Phi) = \sum_{t=0}^{T} \left(\mathbb{E}_d(\phi_t, \psi_t) + \mathbb{E}_s(\phi_t)\right) + \sum_{t=0}^{T-1} \mathbb{E}_n(\phi_t, \phi_{t+1}) \quad (1)$$

Here $\mathbb{E}_d(\phi_t, \psi_t)$ calculates the similarity between ϕ_t and ψ_t, and $\mathbb{E}_n(\phi_t, \phi_{t+1})$ measures how closely Φ satisfies the physically-

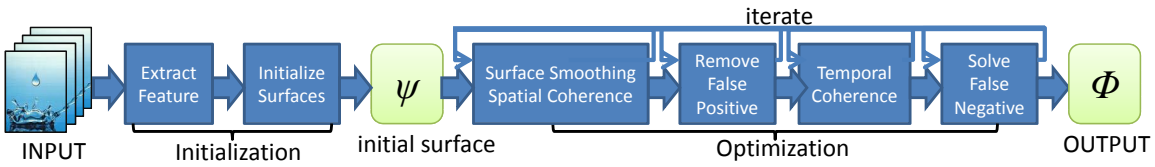

Figure 2: *The entire fluid modeling system pipeline.*

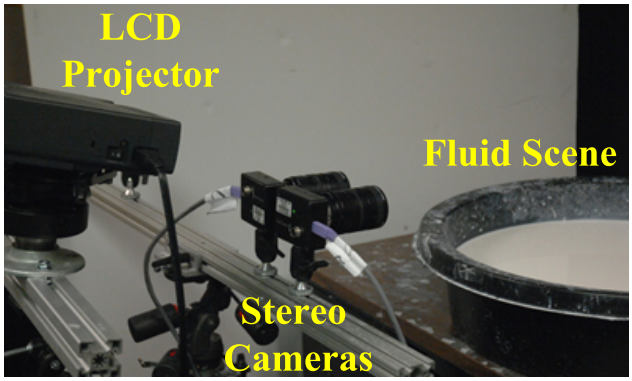

Figure 3: *The capturing setup.*

based constraints in local. Besides spatial-temporal coherence, a smoothness term $\mathbb{E}_s$ is also introduced to provide surface smoothness, which is a common observation in real fluid scenes due to surface tension. In our method, each energy in Equation 1 is treated as an independent sub-problem and solved separately multiple times to obtain a final water animation that reaches a minimum of the sum of all energies (as illustrated in Figure 2). Our experiment indicates that five iterations are sufficient for most optimization cases.

We choose implicit signed distance functions to represent the 3D fluid geometry for two reasons. First, signed distance functions are neutral to frequent topological changes that occur in the evolution of fluids. Second, the 3D flow estimation method can easily take advantage of this representation for comparing similarity between iso-surfaces, as discussed in Section 6.1. Our signed distance functions are discretely defined on a regular grid for data structure simplicity.

4 Surface Initialization

We use a single pair of calibrated cameras for scene acquisition. This consists of two synchronized high-speed greyscale Dragonfly Express cameras. The water is contained in a small round tub, and the water is dyed with white paint, making it opaque to allow light patterns to be projected onto its surface. A LCD projector is positioned just behind the cameras, and it projects a fixed random pattern to provide artificial spatial features on water. The projector position is chosen to minimize shadows, since this would cause spatial features to be lost. We typically capture the scene at a resolution of 640×480 at 200fps.The equipment setup is shown in Figure 3.

4.1 Depth Extraction

Captured stereo sequences $\{I_t\}$ and $\{J_t\}$ are first rectified according to epipolar geometry so that any 3D point will be projected onto the same horizontal line in both I_t and J_t. The depth of pixel p on I_t is determined from its feature correspondence p' on J_t. Since

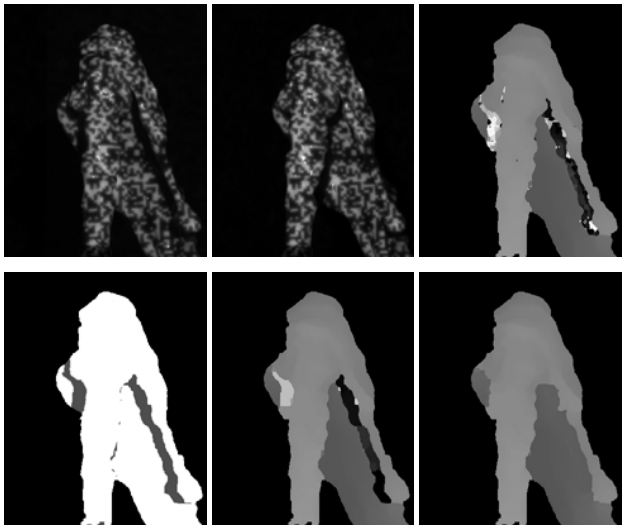

Figure 4: *The first two images in the top row are captured images from stereo camera after rectification. A random texture pattern is projected on the water surface. The top right image is the noisy depth result without belief propagation. The bottom row from left to right shows troubled regions (gray) recognized by the two-pass algorithm, the depth result before and after the two-pass algorithm.*

I_t and J_t are rectified, p' must be on the same horizonal line as p. The problem of stereo matching is to find the *disparity*, which is defined as the difference of horizontal coordinates between p and p'. Stereo is a well studied problem in computer vision. We choose the method in [Sun et al. 2003] which is based on belief propagation. In addition, we use a sub-pixel interpolation scheme proposed in [Yang et al. 2007] to smooth the depth map and avoid aliasing artifacts that are caused by disparity vector discretization.

Regions that are in shadow or occluded from one of the cameras will produce depth errors due to missing correspondences. Those regions are near depth boundaries and they usually belong to the further (occluded) surfaces. Our solution is a two-pass algorithm. In the first pass, a left/right check in [Egnal and Wildes 2002] is carried out over depth maps from both cameras to identify regions with inconsistent disparity values. Then in the second pass, the belief propagation algorithm estimates the depth map as usual without the pixels in troubled regions. After that, their depth values are assigned from already calculated regions through Laplacian smoothing. Figure 4 shows the depth result of a fountain example.

4.2 Surface Initialization

The depth map created in Section 4.1 only covers part of the real fluid surface, in fact, less than 50%. Since the optimization method in Section 6 requires a roughly complete surface as input, we use a heuristic method for creating an initial guess of the surface. All sur-

faces starting from this point on are represented as signed distance functions on a regular volumetric grid.

Using Γ_t, the partial surface defined according to the depth map at time t, an initial surface ψ_t can be created by the union of all spheres centered at Γ_t with radius r:

$$\psi_t(\vec{x}) = \min_{y \in \Gamma_t}(\|\vec{x} - \vec{y}\| - r) \tag{2}$$

This method is effective when the water behaves like a thin shell or film, as our *pouring* example in Figure 12. It is not sufficient when large water regions are missing due to occlusion, therefore, simple heuristics are used in our experiments to approximate the missing region. For instance, the splash example of Figure 14 is nearly symmetric to a vertical plane $z = z_0$. We first fill in the water volume beneath the visible surface as a height field. Let $\vec{y}$ be the lowest point in Γ_t on the same vertical line as a grid cell $\vec{x}$:

$$\vec{y} = \arg\min \vec{y}_y \qquad (\vec{y} \in \Gamma_t, \quad (\vec{y} - \vec{x}) \parallel Y\ axis\ ) \tag{3}$$

For any $\vec{x}$ whose $\vec{y}$ exists, its signed distance value is simply:

$$\psi_t(\vec{x}) = \vec{y}_y - \vec{x}_y \tag{4}$$

If $\vec{x}$ doesn't have $\vec{y}$ but its symmetric counterpart $\vec{x}' = \vec{x} + (2z_0 - \vec{x}_y)(0, 1, 0)^T$ to the plane has a $\vec{y}'$ defined, the signed distance can then be copied from its counterpart as: $\psi_t(\vec{x}) = \psi_t(\vec{x}')$. Finally, the value for the rest undefined grid cells are blended from its four horizontal neighbors to complete the surface ψ_t. An example is shown in Figure 5.

Space-Time Merging When a water scene exhibits temporal repetitiveness rather than spatial repetitiveness, similar ideas can be applied to reconstruct the initial shape ψ_t using video sequences captured at different time from different viewpoint. For example, a single pair of cameras is unable to simultaneously capture both the downstream (front) and the upstream (back) in our fountain case in Figure 6 due to occlusions. Our solution is to capture each of these streams in turn, e.g., first capture the upstream view, than move the camera around to capture the downstream view. While each surface immediately generated from video cannot cover the whole fluid surface, they can be stitched together assuming that the temporal repetition gives a similar dynamic appearance over time. First the relative camera pose of each view can be found using static points (fiducials) in the scene. From the pose information, different sequences can be aligned. Note that this alignment does not require high-accuracy, even slightly moving points can be used as fiducials. If original sequences are sufficiently long, we choose two optimal sub-sequences from the original input such that the overall difference between them are minimized after spatial alignment. The shape difference is measured as the sum-of-squared difference between two signed distance functions. Although the resulting surface ψ_t may not correspond exactly to a real water scene, our experiment shows that this approach produces visually plausible results.

5 Surface Smoothing and Spatial Coherence

Real fluid surfaces are smooth in space due to the effect of surface tension, especially for water at small scales. To provide similar results in our method, we use a surface smoothing scheme each time the surface sequence is updated.

Mean curvature flow can provide a surface smoothing effect, and this can be achieved using the following level set formulation:

$$\phi_s = \kappa \cdot |\nabla \phi| \tag{5}$$

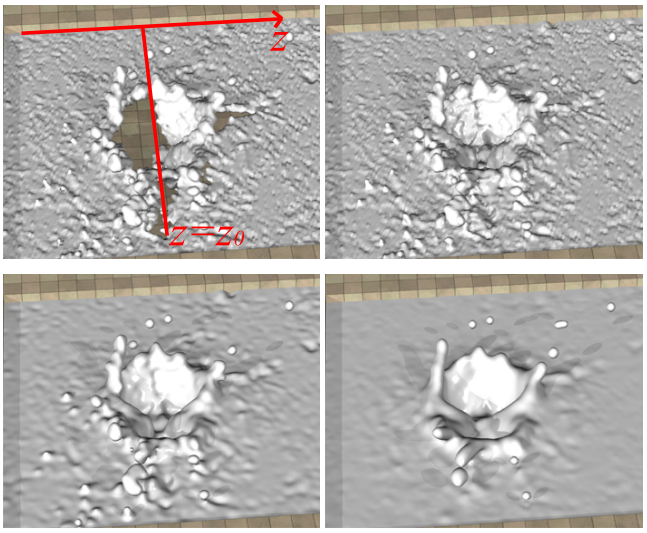

Figure 5: *The initialization result for the splash example. From left to right and top to bottom are the partial surface Γ_t from the depth map, surface after initialization, surface after initialization and smoothing, and the final reconstructed surface respectively.*

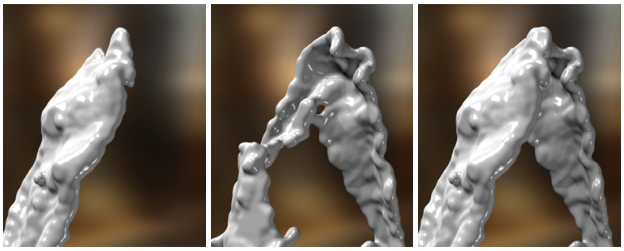

Figure 6: *While each surface reconstructed solely in the front view (left) or back view (middle) cannot cover the whole surface, a reasonable initial surface can be generated by assembling two views together, even if they are captured at different time.*

in which $s \to \infty$, standing for a steady solution. κ is the surface mean curvature.The mean curvature flow tries to minimize the surface area by making mean curvature uniform over the whole surface, so that the tendency is for the surface to evolve towards a spherical shape. Using such a level-set mean curvature evolution is straightforward to implement, but it performs excessive smoothing and does not take into account fidelity to the original geometry (our spatial constraints). In fact, although the surface tension effect is similar to the mean curvature flow over time, temporal surface tension distribution on a time-varying surface is not the steady state of the mean curvature flow. Instead, we choose the smoothing scheme proposed by Schneider and Kobbelt [2001] under the following target PDE:

$$\Delta_B \kappa(\vec{x}) = 0 \tag{6}$$

Δ_B is the Laplacian-Beltrami operator. Assuming that κ along the surface normal over iso-surfaces is locally linear, Δ_B can be approximated by a Laplacian operator in 3D, and the solution is an evolution equation in fourth order:

$$\phi_s = \Delta\kappa \cdot |\nabla \phi| \tag{7}$$

Intuitively, this will produce a fluid surface with C^2 continuous mean curvature, leading to a C^2 surface tension field on the surface.

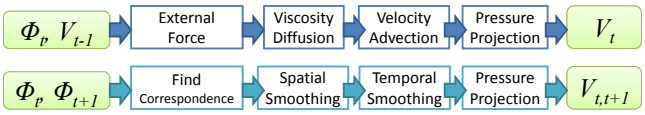

Figure 7: *The velocity field estimation scheme for physically based fluid simulation (top) and the 3D flow estimation method (bottom).*

This fourth-order equation is better at preserving spatial constraints, and it also preserves volumes according to the divergence theorem. Equation 7 is solved over time iteratively using a first-order forward Euler method. Instead of calculating $\Delta\kappa$ directly using a four-order scheme, we first calculate κ for each grid cell close to the surface boundary, then calculate $\Delta\kappa$, both by central differencing:

$$\Delta\kappa = \nabla \cdot \nabla\kappa = \kappa_{xx} + \kappa_{yy} + \kappa_{zz} \tag{8}$$

In practice, we combine the surface smoothing step together with a fidelity term (penalizing differences from the input geometry):

$$\phi_s = \alpha(\psi - \phi) + \Delta\kappa \cdot |\nabla\phi| \tag{9}$$

α is a coefficient balance between fidelity and smoothing strength, ranging from 0 to 0.1. For the data sets we have processed, only 5 to 10 iteration steps are needed to produce good smoothing results.

6 Surface Optimization

In this section we discuss our approach for enforcing temporal coherence, which corresponds to $\mathbb{E}_n$ in Equation 1. We first introduce a 3D temporal flow estimation method in Section 6.1. We then discuss our optimization method that generates surface sequences by interpolating shapes in neighboring frames. This optimization method cannot be applied on input sequence Ψ directly due to local minimum caused by two types of errors: false-positives, when certain surface components fail to find their temporal partners in adjacent frames since the component does not exist in the real fluid scene; and false-negatives, when real fluid regions are missed, preventing other frames from finding temporal correspondences. Therefore, two extra steps will be used to address those issues in Section 6.2 and 6.4, respectively.

6.1 3D Temporal Flow Estimation

Given a fluid surface ϕ_t at time t and a velocity field $\vec{v}_{t-1,t}$ from the previous time-step, physically based fluid simulation first finds the new velocity flow $\vec{v}_{t,t+1}$ by solving the fluid dynamics equations, then generates ϕ_{t+1} by tracing ϕ_t along $\vec{v}_{t,t+1}$. Different from fluid simulation, both ϕ_t and ϕ_{t+1} are assumed to be given as input to our problem, and the goal in this section instead is to find the corresponding velocity field $\vec{v}_{t,t+1}$ that is most likely to cause the evolution between the two surfaces.

The dynamic behavior of water can be described by incompressible viscous fluid dynamics according to the Navier-Stokes equations. Most fluid simulation methods in graphics use operator splitting to calculate the separate terms of the Navier-Stokes equations. In such cases, the fluid solver first applies external forces on $v_{t-1,t}$, then advects the velocity flow by itself, damps the velocity flow using viscosity diffusion, and finally projects it back to a divergence-free space to maintain incompressibility. Figure 7 top shows these steps. The resulting velocity flow $\vec{v}_{t,t+1}$ not only evolves from ϕ_t to ϕ_{t+1}, but also satisfies the following properties: 1) temporal continuity due to velocity advection, 2) spatial continuity due to viscosity diffusion, and 3) incompressibility due to pressure projection. Similarly, our flow estimation method will also consider these properties, as shown in Figure 7 bottom. Our method first enforces the shape evolution constraint by finding 3D correspondences between ϕ_t and ϕ_{t+1}. This initial step also implicitly includes external forces and temporal continuity constraints. We then smooth $\vec{v}_{t,t+1}$ in the spatial domain and finally enforces incompressibility, similar to the viscosity diffusion and pressure projection steps in fluid simulation. This flow estimation problem is related to the classic optical flow problem in computer vision, in which both 2D correspondence constraints and smoothness constraints are satisfied. Our problem is different, however, in that it is defined in space+time and more constraints in addition to spatial smoothness need to be satisfied.

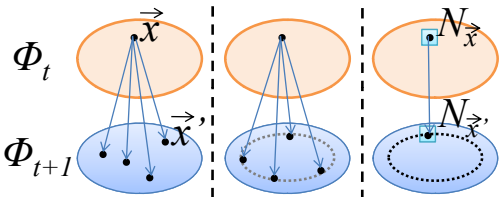

Figure 8: *Three different correspondence methods. For each point $\vec{x}$ in water ($\phi_t(\vec{x}) < 0$), the left scheme matches $\vec{x}$ with any point $\vec{x}'$ in water $\phi_{t+1}(\vec{x}')$. The middle scheme matches $\vec{x}$ with $\vec{x}'$ so that their signed distance values are the same $\phi_t(\vec{x}) = \phi_{t+1}(\vec{x}')$. The right scheme looks for $\vec{x}'$ so that their neighborhoods are similar.*

Correspondence Search A conservative approach to the shape evolution constraint is to find $\vec{v}_{t,t+1}(\vec{x})$ for each water grid cell $\vec{x}$ ($\phi_t(\vec{x}) < 0$) such that $\phi_{t+1}(\vec{x} + \Delta t \cdot \vec{v}_{t,t+1}(\vec{x})) < 0$. Intuitively, this means each water cell at time t should be mapped to a water cell at time $t+1$. Unfortunately, there are many ambiguities if this method is used, as illustrated in Figure 8 left. Instead, we can search for correspondence with the same signed distance value $\phi_{t+1}(\vec{x} + \Delta t \cdot \vec{v}_{t,t+1}(\vec{x})) = \phi_t(\vec{x})$, and this successfully limits the ambiguities to being only on iso-surfaces, as shown in Figure 8 middle. To further reduce ambiguities, our method uses a local $3 \times 3 \times 3$ grid neighborhood $N_{\vec{x}}$ centered at $\vec{x}$ to represent the shape feature. We ask that the estimated velocity takes $\vec{x}$ and its neighbors to closely matched signed distance values:

$$\begin{gathered}\vec{v}_{t,t+1}(\vec{x}) = \arg\min \sum_{\vec{y}\in N_{\vec{x}}} w(\vec{y}) \left(\phi_{t+1}(\vec{y}') - \phi_t(\vec{y})\right)^2 \\ \vec{y}' = \vec{y} + \vec{v}_{t,t+1}(\vec{x})\Delta t\end{gathered} \tag{10}$$

$w(\vec{y})$ is a weight function for the neighborhood, typically a Gaussian falloff. Finally, the opposite direction from ϕ_t to ϕ_{t-1} is also included to reflect external forces and velocity advection between velocity flows at different time:

$$\begin{gathered}\vec{v}_{t,t+1}(\vec{x}) = \arg\min D(\vec{x}, \vec{v}) = \arg\min \\ \sum_{\vec{y}\in N_{\vec{x}}} w(\vec{y}) \left(\left(\phi_{t+1}(\vec{y}') - \phi_t(\vec{y})\right)^2 + \left(\phi_{t-1}(\vec{y}'') - \phi_t(\vec{y})\right)^2\right) \\ \vec{y}'' = \vec{y} - \vec{v}_{t,t+1}(\vec{x})\Delta t + \vec{a}(\vec{x})\Delta t^2\end{gathered} \tag{11}$$

in which $\vec{a}(\vec{x})$ is the external acceleration at $\vec{x}$. $\vec{y}'$ and $\vec{y}''$ are $\vec{x}$'s correspondence location in ϕ_{t+1} and ϕ_{t-1} respectively. By differentiating $D(\vec{x}, \vec{v})$ with respect to $\vec{v}_{t,t+1}(\vec{x})$, we obtain:

$$\sum_{\vec{y}\in N} w(\vec{y}) \begin{pmatrix} +\left(\phi_{t+1}(\vec{y}') - \phi_t(\vec{y})\right)\nabla\phi_{t+1}(\vec{y}') \\ -\left(\phi_{t-1}(\vec{y}'') - \phi_t(\vec{y})\right)\nabla\phi_{t-1}(\vec{y}'') \end{pmatrix} = 0 \tag{12}$$

$\nabla\phi$ is always close to 1 by the definition of a signed distance function, therefore, Equation 12 can be solved iteratively using the

Newton-Ralphson method. $\vec{v}(\vec{x})$ can be initialized with zeros, several candidate seeds from heuristics, or user estimation if available. For each iteration, a new velocity flow $\vec{v}_{\text{new}}(\vec{x})$ is calculated from the old one $\vec{v}_{\text{old}}(\vec{x})$:

$$\sum_{\vec{y}\in N} w(\vec{y}) \begin{pmatrix} +(\phi_{t+1}(\vec{y}'_{\text{new}}) - \phi_t(\vec{y}))\,\nabla\phi_{t+1}(\vec{y}'_{\text{old}}) \\ -(\phi_{t-1}(\vec{y}''_{\text{new}}) - \phi_t(\vec{y}))\,\nabla\phi_{t-1}(\vec{y}''_{\text{old}}) \end{pmatrix} = 0 \tag{13}$$

$\vec{y}'_{\text{new}}, \vec{y}''_{\text{new}}$ and $\vec{y}'_{\text{old}}, \vec{y}''_{\text{old}}$ are $\vec{y}$'s correspondences calculated using new and old velocity flows respectively. We further linearize $\phi_{t+1}(\vec{y}'_{\text{new}})$ and $\phi_{t-1}(\vec{y}''_{\text{new}})$ as:

$$\begin{aligned} \phi_{t+1}(\vec{y}'_{\text{new}}) &\approx \phi_{t+1}(\vec{y}'_{\text{old}}) + \nabla\phi^T_{t+1}(\vec{y}'_{\text{old}})(\vec{v}_{\text{new}} - \vec{v}_{\text{old}})\Delta t \\ \phi_{t-1}(\vec{y}''_{\text{new}}) &\approx \phi_{t-1}(\vec{y}''_{\text{old}}) - \nabla\phi^T_{t+1}(\vec{y}''_{\text{old}})(\vec{v}_{\text{new}} - \vec{v}_{\text{old}})\Delta t \end{aligned} \tag{14}$$

Combining Equation 13 with 14 gives us a linear system with $\vec{v}_{\text{new}}(\vec{x})$ as unknowns. Iterations are terminated if maximum iteration number is reached or if $\vec{v}_{\text{new}} - \vec{v}_{\text{old}}$ drops below certain threshold (0.05 of a grid cell size in our experiments). We also limit $|\vec{v}_{\text{new}} - \vec{v}_{\text{old}}|$ by an upper bound for smoother convergence. We also limit the search space within a range around the initial guess to prevent large velocity changes in case the problem is ill-conditioned. This solution is similar to the classic Kanade-Lucas-Tomasi (KLT) feature tracker [Shi and Tomasi 1994] with translation motions only, except that our problem is defined in 3D and the search procedure considers both forward and backward directions.

When two separate water regions merge or split, this estimation scheme may not work for all grid cells. For instance, a grid cell may not find its actual correspondence according to the neighborhood similarity in Equation 11, when it represents a water drop dripping into a still water surface. Similar to the aperture problem in KLT, this scheme also fails to work when the fluid shape barely changes, i.e., a static flow. Fortunately, both problems can be solved using smoothness and incompressibility constraints by propagating correctly estimated flows into ill-defined regions, as will be discussed next. Figure 9 shows a frame in a water splash example and its estimated velocity field using this algorithm.

If the correspondence search in one direction goes beyond the spatial-temporal boundary, we assume that everything is possible outside of the 4D space-time grid, and the corresponding error in Equation 11 and correction in 13 are simply ignored.

Error Measure $D(\vec{x}, \vec{v})$ in Equation 11 is used as an error measure to determine whether $\vec{x}$'s correspondence exists in the neighboring frame. If D is below some tolerance ϵ at time t, we categorize $\vec{x}$ at time t as Type-I, which means it has correspondences in both ϕ_{t-1} and ϕ_{t+1}. Otherwise, the method will try to locate correspondences only in ϕ_{t-1} or ϕ_{t+1}, with half of the tolerance $\epsilon/2$. We do this by removing the error and correction contribution from the other neighboring frame in Equation 11 and 13. Grid cells will be type-II if correspondence exists in ϕ_{t+1}, or type-III, if correspondence exists in ϕ_{t-1}. The rest of grid cells will be type-O, which means no correspondence is found in either ϕ_{t-1} or ϕ_{t+1}.

Spatial Smoothing The spatial smoothing step here mimics the viscosity effect, similar to using an explicit solver by discretizing the Laplacian operator for viscosity effects:

$$\vec{v}_{t,t+1}(\vec{x}) = \sum s(\vec{y})\vec{v}_{t,t+1}(\vec{y}) \tag{15}$$

$\vec{y}$ is a water grid cell within an immediate grid neighborhood of $\vec{x}$, including $\vec{x}$ and its six neighbors. Typically, $s(\vec{y})$ defines the spatial smoothing kernel covering $\vec{p}$ and its six immediate neighbors as:

$$s(\vec{y}) = 1 - 6\beta \quad (\vec{y} = \vec{x}), \qquad s(\vec{y}) = \beta \quad (otherwise) \tag{16}$$

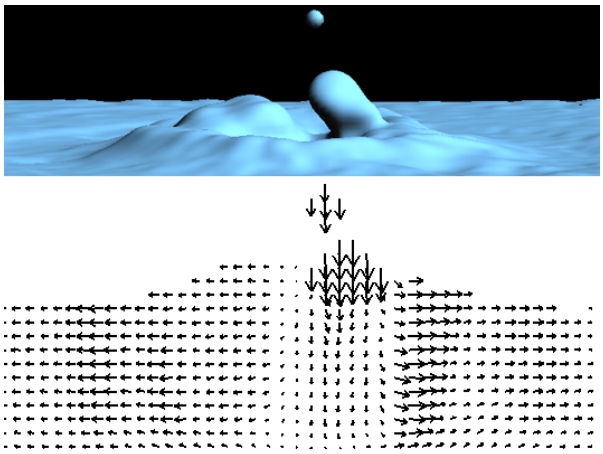

Figure 9: *One frame in a water splash example and its estimated velocity field visualizated as a vector diagram.*

β is specified by user according to their expectation of velocity spatial smoothness. It should be noted that although this method looks similar to the explicit viscosity solver in physically based fluid solver, it only provides similar effect as viscosity diffusion and it is not based on actually fluid dynamics since only a static velocity field is involved.

To generate large viscosity effect, some fluid simulation algorithms [Stam 1999; Carlson et al. 2002] formulate viscosity diffusion implicitly as a sparse linear system. Our method chooses to simply apply the smoothing process multiple times to produce a visually plausible velocity flow.

Pressure Projection While the velocity is defined at each grid cell for the convenience of estimation and optimization, it is not straightforward to couple velocity with pressure, which is also defined at the grid cell. Our simple solution is to first interpolate the velocity flow to a staggered grid formulation by averaging, then apply pressure projection, and convert the velocity back to the grid cell center. Details about the pressure projection in a staggered Mark-and-Cell (MAC) grid can be found in [Stam 1999]. We use Dirichlet boundary condition with zero outer pressure when water bodies are completely within the space-time grid. Water bodies expanding beyond the space-time grid are ignored in this step since their volumes outside of the grid are unknown.

After the pressure projection step, error scores are recalculated and grid cells are re-categorized. This category information will be used to determine whether missing correspondence is caused by false-positives or false-negatives.

6.2 False-Positive Removal

Since both false-positives and false-negatives can cause missing correspondences in neighboring frames, the first task is to determine the cause of a missing correspondence. We do this by counting how many consecutive frames a water region appears in, assuming that real water should at least exists in several consecutive frames. For example, when a water region only appears in frame t, or frame t and $t+1$, it is less likely to be a water component in the real scene. However, if it exists in more frames, missing correspondences are more likely to be caused by false-negatives, so the component should not be removed. This assumption fails in certain cases, for example, a water drop intermittently missing in a

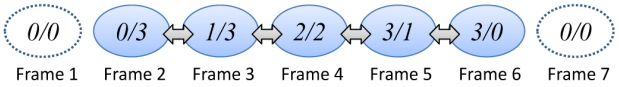

Figure 10: *Forward/backward tags for a water drop that sequentially appears from frame 2 to 6.*

sequence will be treated as a false-positive even though it is more likely to be in the real scene. Fortunately, our experiment shows that such cases are rare and both false-positives and false-negatives are usually distributed continuously over time, so this assumption can be safely held.

To count how many times a water region appears in previous and successive frames, two count tags are assigned to each water grid cell: a forward tag T_t^f and a backward tag T_t^b. Here we will explain how to set up T_t^f by tracing particle from each grid cell. Backward tags are calculated in a similar fashion, except in the opposite direction. All T_t^f are initialized to be zero, and then processed forward from frame 0 to frame T. In each frame, every type-I water grid cell $\vec{x}$ first increases its T_t^f by 1, and propagates its value to all eight cells adjacent to $\vec{x}' = \vec{x} + \vec{v}_{t,t+1}(\vec{x})\Delta t$ in frame $t+1$:

$$T_{t+1}^f(\vec{y}) = \max(T_t^f(\vec{x}), T_{t+1}^f(\vec{y})), \quad \vec{y} \approx \vec{x}' \tag{17}$$

Figure 10 shows a single water drop example with both tags calculated. $T_t^f + T_t^b + 1$ is the number that a water grid cell appears in previous and successive frames, and $T_t^f + T_t^b + 2$ is the number in two ends. For each water grid cell $\vec{x}$ at time t, if $T_t^f + T_t^b$ is below some threshold k, it is considered as false-positive and will be removed by setting $\phi_t(\vec{x})$ to infinity. Theoretically, the trajectory of a free-falling water region existing in three consecutive frames can be sufficiently predicted, assuming that the acceleration is constant. In practice we choose k from 3 to 7, in order to remove more false-positives and keep more confidence in remaining water regions.

The ambiguity in velocity estimation may prevent false-positives from be removed as shown in Figure 11 top. To solve this issue, we compare $\vec{v}_{t,t+1}(\vec{x})$ with $\vec{v}_{t+1,t+2}(\vec{x}')$, and if their difference is above certain threshold θ, $\vec{x}'$ will be considered to involve velocity ambiguity, tagged as negative, and will lose its ability to propagate. One may ask whether a temporal smoothing step can be introduced into the velocity estimation method instead so water region in frame 4 can choose the right velocity flow through exchanging velocity with its neighbors. Unfortunately it is impossible to know which velocity is correct until the false-negatives and false-positives are recognized, and we have to defer such decisions until the false-positive removal step.

The pseudo code for computing forward tags is described in Algorithm 1. The pouring example in Figure 12 shows redundant water regions are removed as false-positives by this algorithm.

6.3 Optimization for Temporal Coherence

After the false positives have been removed, we can increase the quality of between-frame coherence using the estimated flow velocity. Let $\vec{x}$ be a type-I grid cell with both correspondences in the space-time grid. $\mathrm{P}^-(\phi_{t-1}, \vec{x}, \vec{v}_{t-1,t})$ and $\mathrm{P}^+(\phi_{t+1}, \vec{x}, \vec{v}_{t,t+1})$ are two functions that predict $\vec{x}$'s signed distance value according to neighboring frames ϕ_{t-1} and ϕ_{t+1} respectively, and the solution is to iteratively refine ϕ_t by interpolating its predicted values:

$$\begin{aligned}\phi_t(\vec{x}) = \gamma(&\mathrm{P}^-(\phi_{t-1}, \vec{x}, \vec{v}_{t-1,t}) + \mathrm{P}^+(\phi_{t+1}, \vec{x}, \vec{v}_{t,t+1})) \\ &+(1-2\gamma)\phi_t(\vec{x})\end{aligned} \tag{18}$$

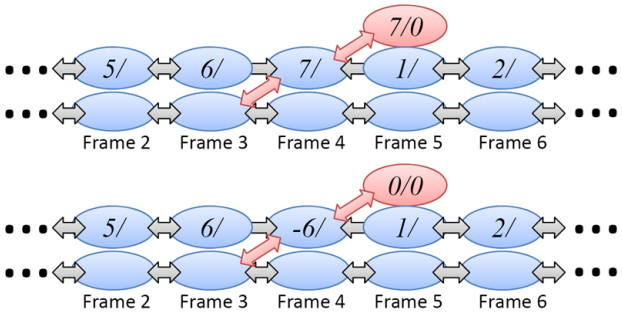

Figure 11: *When two water drops (in blue) sequentially appear from frame 2 to 6 and an incorrect water drop (in red) suddenly appears in frame 5, their particular positions cause an ambiguity in the velocity flow at frame 4. Without the velocity consistency check at the top diagram, T_4^f will be erroneously propagated to the red water drop, preventing it from being removed. After the consistency check, this can be addressed as shown in the bottom diagram.*

Algorithm 1 The pseudo code for calculating forward tags.

```
for t = 0 to T do
  T_t^f = 0;
end for
for t = 0 to T do
  for each fluid cell x⃗ do
    if φ_t(x⃗) < band_width and type(x⃗, t) = I then
      if T_t^f(x⃗) < 0 then
        T_t^f(x⃗) = 0;
      end if
      T_t^f(x⃗) += 1;
      next = T_t^f(x⃗);
      x⃗' = x⃗ + v⃗_{t,t+1}(x⃗)Δt;
      if |v⃗_{t,t+1}(x⃗) − v⃗_{i+1,i+2}(x⃗')| > θ then
        next = −next;
      end if
      T_{t+1}^f(y⃗) = max(next, T_{t+1}^f(⌈x'_x⌉, ⌈x'_y⌉, ⌈x'_z⌉));
                  .....
      T_{t+1}^f(y⃗) = max(next, T_{t+1}^f(⌊x'_x⌋, ⌊x'_y⌋, ⌊x'_z⌋));
    end if
  end for
end for
```

in which γ is a relaxation coefficient between 0 and 0.5. Since the optimization step at time t only involves three frames ϕ_t, ϕ_{t-1} and ϕ_{t+1} as in Equation 18, and the flow estimation process of $\vec{v}_{t,t+1}$ already considers three frames in both directions, we replace $\vec{v}_{t-1,t}$ by $\vec{v}_{t,t+1} - \vec{a}\Delta t$ for better local temporal consistency,

$$\begin{aligned}\phi_t(\vec{x}) = \gamma(&\mathrm{P}^-(\phi_{t-1}, \vec{x}, \vec{v}_{t,t+1} - \vec{a}\Delta t) \\ &+\mathrm{P}^+(\phi_{t+1}, \vec{x}, \vec{v}_{t,t+1})) + (1-2\gamma)\phi_t(\vec{x})\end{aligned} \tag{19}$$

$\vec{a}$ is user-specified external acceleration, if applicable. In fact, using $\vec{v}_{t-1,t}$ in Equation 18 would allow ϕ_{t-2} to have unnecessary influence over the optimization of ϕ_t, which should be avoided.

To implement a prediction function, one may think of evolving ϕ_{t-1} and ϕ_{t+1} to time t using the given velocity flow using by the level set method. This implementation is limited by the CFL condition, and must be done in multiple sub-steps when the fluid surface moves much more than one grid cell in distance. To avoid the computational cost incurred in subdividing time steps, a simple alternative is to use the signed distance value of $\vec{x}$'s correspondence

in both frames:

$$\mathrm{P}^{+}(\phi_{t+1}, \vec{x}, \vec{v}_{t,t+1}) = \phi_{t+1}(\vec{x} + \vec{v}_{t,t+1}\Delta t) \tag{20}$$

similar to a simplified one-step semi-Lagrangian method. Unfortunately, this method suffers from volume lose because the required resampling process gradually smears out the surface sequence in each iteration. Instead, we use a non-interpolating semi-Lagrangian method described in [Staniforth and Côté 1991] by combining it with the level set method, under the assumption that the velocity flow is locally uniform due to viscosity effects. This method first separates a displacement vector $\vec{v}_{t,t+1}(\vec{x})\Delta t$ into a rounded-up integer component $\vec{o}_i$ and the remaining floating component $\vec{o}_f$. The surface motion in the integral displacement is to shift signed distances in $\vec{x}$'s local neighborhood to $\vec{x} + \vec{o}_i$. The remaining surface motion corresponding to $\vec{o}_f$ is less than half of a grid cell in any axis, so the level set method can be safely used without stability issues. This method is described as follows:

$$\begin{gathered} \mathrm{P}^{+} = \phi_{t+1}(\vec{x} + \vec{o}_i) - (-\vec{o}_f)\nabla\phi_{t+1}(\vec{x} + \vec{o}_i) \\ \vec{o} = \vec{v}_{t,t+1}\Delta t \\ \vec{o}_i = \mathrm{Round}(\vec{o}); \quad \vec{o}_f = \vec{o} - \vec{o}_i \end{gathered} \tag{21}$$

When the surface motion is originally less than half of a grid cell ($\vec{o}_i = 0$), the method is simply reduced to the level set method. This hybrid method can successfully reduce volume loss to less than 5% over more than 5 optimization iterations.

The above scheme is for type-I grid cells. If a grid cell $\vec{x}$ belongs to type-II or III, or if one of $\vec{x}$'s correspondence goes beyond the space-time grid, it is impossible to predict from both directions. Instead, we only consider one direction and the following equation calculates the update using only the previous prediction:

$$\phi_t(\vec{x}) = \gamma\mathrm{P}^{-}(\phi_{t-1}, \vec{x}, \vec{v}_{t-1,t}) + (1-\gamma)\phi_t(\vec{x}) \tag{22}$$

6.4 False-Negative Completion

The final step in the system pipeline is to resolve false negatives by adding missing water regions back. Type-II and III grid cells can be safely assumed to be caused by false-negatives at this time. We first propagate their velocities to the frame with no correspondence by linear resampling. We then apply the one-way prediction tool provided in Equation 22 to grid cells with newly updated velocities to recover missing water regions. The false-negative completion process is executed more than once to allow water regions to propagate over multiple frames. After this, the whole optimization procedure may be performed again to remove any temporal-spatial incoherence caused by false-negative completion.

7 Results and Discussion

In this section we present results from several different real-world fluid examples. (Please watch the accompanying video to see animations for each of these results.) The scene in the pouring example shown in Figure 12 is made of free-falling streams poured out of a coffee cup. The animation is reconstructed at a resolution of $176 \times 120 \times 120$ for 155 frames. The splash example in Figure 14 shows the scene of a bottle lid dropping into an open water area. The resolution of the 3D volume grid in this example is $158 \times 120 \times 163$ and includes 138 frames. Since it is difficult to discriminate the lid from the water surface, we modeled the lid as part of the fluid surface as well. External acceleration and incompressibility are ignored for the large water body in the pool because it expands beyond the spatial grid and the pressure projection step cannot be easily achieved for it. The fountain example in Figure 1 (on the first page) was reconstructed from two video sequences

Table 1: *Common variables and their values in our experiment.*

Name	Definition	Value
α	a balancing coefficient between fidelity and shape smoothness in Equation 9	0.1
β	a parameter used in the velocity smoothing kernel in Equation 16	$1/7$
γ	a relaxation coefficient for temporal coherence by interpolation in Equation 18	0.3
ϵ	a tolerance bound in **Error Measure**, in Section 6.1	$[64, 256]$
k	a threshold to determine false positives by tag values in Section 6.2	$[3, 7]$
θ	a threshold to avoid velocity inconsistency when calculating tags in Section 6.2	0.2

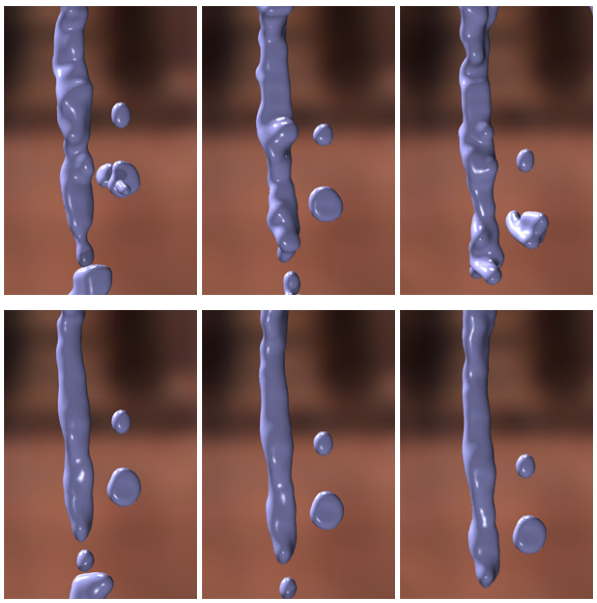

Figure 12: *The pouring example: The top row shows the result without using any surface optimization. The bottom row show the result after applying our algorithm.*

captured at different times from different viewpoints, containing the upstream and the downstream views of a small water fountain by placing stereo camera in front and back respectively. The resolution is $110 \times 150 \times 140$ and the result has 120 frames.

All examples are calculated on an Intel Quad-Core 9550 workstation with 8G memory. The initial reconstruction and smoothing process in the first part of our algorithm took 10-30 mins typically. Velocity estimation took 20-40 mins and the rest of the surface optimization algorithm took 20-30 mins. Five surface optimization iterations are usually sufficient to generate acceptable results. Overall, each frame takes 2 to 3 mins to process on average. Compared with physically-based fluid simulation, this method is not limited to any CFL condition and the surface motion in each time step ranges between 4 to 8 grid cell sizes on average. At the same resolution, a physically-based simulation would have to use smaller time steps to maintain stability and accuracy, causing significantly higher computational cost than this method, especially for rapid water motion.

Memory becomes an issue in our experiments since loading the whole 3D volume sequence with the velocity field and other data structure would require at least 6GB of memory. One possible solu-

tion would be to create a dynamic grid data structure that allocates memory only to grid cells close to the water surfaces. Here we choose to keep a simple grid data structure and use a large amount of virtual memory instead. Each iteration procedure requires at least one full memory swap to disk for each frame and contributes about 10% to 20% of the computational time.

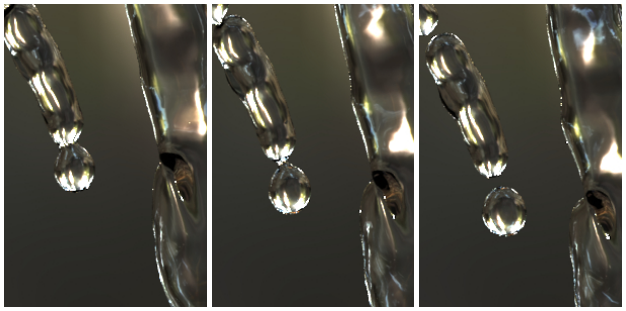

Figure 13: *The middle shape is interpolated from the left shape and the right shape according to the estimated velocity field. In this way, water drops break up continuously from the stream.*

Space-Time Interpolation and Extrapolation The estimated velocity flow can be used to increase the temporal resolution of a sequence or to complete missing frames in case of data capture failure. Figure 13 shows that the shape in the middle frame is successfully interpolated from the left shape and the right shape. In addition, our algorithm can be considered as a feature-based morphing method even without requiring shape alignment as pre-processing. When treating regions beyond the space-time grid as missing regions in the false-negative completion method in Section 6.4, our algorithm can also automatically extrapolate shapes in those regions in a spatial-temporal fashion as shown in Figure 15.

Discussion Our experiments show that our results can faithfully inherit the nuances and details of the fluids from regular video input. Our optimization algorithm converges in less than five iterations, therefore, it is safe from error accumulation over time. For example, volume changes are barely noticeable in most cases even though volume preservation is not strictly enforced in our algorithm. Since the problem is under-constrained and the temporal coherence assumption may not necessarily be true when the flow is under strong acceleration, a sudden splash wave for example, the generated result may not be identical to the real water scenes. On the other hand, the solution is a local-minimum that well satisfies spatial-temporal coherence, and provides a convincing and visually plausible result for graphics applications.

Our current framework depends on depth from stereo and heuristic surface initialization algorithms to provide a good initial estimate for the surface optimization process. To improve the stereo depth results, our existing capturing system uses dyed water and a projected pattern for data acquisition. Therefore it is still not possible to capture water outside a studio setting, though we have dramatically reduced the hardware requirement to a single pair of cameras and a fixed projection pattern. Video-based acquisition can provide information only for visible surfaces, we have to use heuristics to estimate the volume. When occlusion is severe, the estimated volume may not be correct. In the fountain example, the water flow appears to be too thin when viewed from a side angle. Using multiple cameras can ameliorate this problem, but does not fully solve it. Under the single-stereo-camera paradigm, user interaction may be the only way to address this problem.

Figure 15: *Water regions beyond the spatial grid (in red) can be easily generated by spatial-temporal extrapolation. We begin with data only in the small vertical region shown at the left, and we create a final sequence with a larger vertical component (right).*

8 Conclusion and Future Work

We have presented a hybrid framework to efficiently reconstruct realistic water animation from real water scenes by using video-based reconstruction techniques together with physically-based surface optimization. Using depth maps from stereo vision, the novelties of our framework include a surface initialization algorithm to create a rough initial guess of the surface sequence, a 3D flow estimation method to define temporal coherence, and a surface optimization method to remove false-positives, enforce temporal coherence and complete false-negatives.

Looking towards the future, a more robust acquisition system and algorithms that do not require the dying of water is our first priority. Solving this will make it practical to capture the complex shapes and interactions of real water in our everyday environment, not just in a laboratory. Since our method does not require feature tracking in video, we are optimistic that some active sensing methods may be developed. In addition, we are also interested in finding a more memory efficient data structure for the grid representation and in creating a GPU implementation of our algorithm for acceleration. We speculate that there may be a way to more closely combine the physically-based optimization and video-based reconstruction. We are planning to explore other interesting fluid effects, including viscoelastic fluid and gas animation. Providing more flexible tools based on this algorithm is also an interesting topic that we plan to study in the future, and this will help artists to design specific water animation effects via editing the end visual images directly, rather than setting up initial conditions and various force controls.

More broadly, we believe this combination of reconstruction with simulation can be extended to model many dynamic real-world objects with the promise of *significantly* reducing the amount of captured samples. Ultimately we hope that the acquisition of 4D models can be as simple as sweeping a camera around – something currently limited to static scenes only. Adding simulation into the pipeline is, in our view, a very promising direction.

Acknowledgements

We would especially like to thank Minglun Gong, kun Zhou, and Howard Zhou for various useful suggestions in this project. Also, we would like to thank everyone who spent time on reading early versions of this paper, including the anonymous reviewers. This research is supported in part by NSF grants CCF-0625264, HCC-0448185, CPA-0811647 and CCF-0811485.

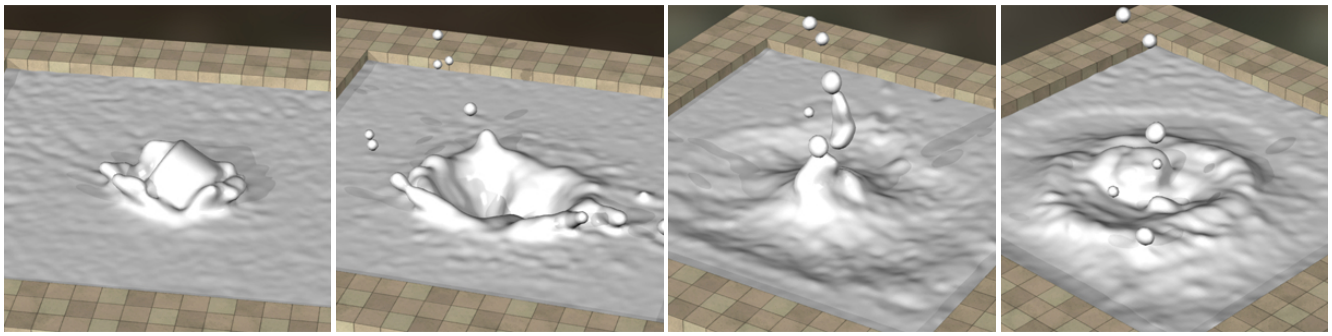

Figure 14: *The splash example.*

References

ATCHESON, B., IHRKE, I., HEIDRICH, W., TEVS, A., BRADLEY, D., MAGNOR, M., AND SEIDEL, H.-P. 2008. Time-resolved 3d capture of non-stationary gas flows. In *Proc. of ACM SIGGRAPH Asia 2008*, vol. 27.

BHAT, K. S., SEITZ, S. M., HODGINS, J. K., AND KHOSLA, P. K. 2004. Flow-based video synthesis and editing. In *Proc. of ACM SIGGRAPH 2004*, 360–363.

BRADLEY, D., POPA, T., SHEFFER, A., HEIDRICH, W., AND BOUBEKEUR, T. 2008. Markerless garment capture. In *Proc. of ACM SIGGRAPH 2008*, vol. 27.

CARLSON, M., MUCHA, P. J., VAN HORN III, R. B., AND TURK, G. 2002. Melting and flowing. In *Proc. of SCA '02*, 167–174.

DE AGUIAR, E., STOLL, C., THEOBALT, C., AHMED, N., SEIDEL, H.-P., AND THRUN, S. 2008. Performance capture from sparse multi-view video. In *Proc. of ACM SIGGRAPH '08*, 1–10.

EGNAL, G., AND WILDES, R. P. 2002. Detecting binocular half-occlusions: Empirical comparisons of five approaches. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 8, 1127–1133.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. In *Proc. of ACM SIGGRAPH '02*, 736–744.

FATTAL, R., AND LISCHINSKI, D. 2004. Target-driven smoke animation. In *Proc. of ACM SIGGRAPH 2004*.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proc. of SIGGRAPH '01*, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graph. Models Image Process. 58*, 5.

GRANT, I. 1997. Particle image velocimetry: a review. In *Proc. of the Institution of Mechanical Engineers*, vol. 211, 55C76.

HAWKINS, T., EINARSSON, P., AND DEBEVEC, P. 2005. Acquisition of time-varying participating media. In *Proc. of ACM SIGGRAPH 2005*.

HULLIN, M. B., FUCHS, M., IHRKE, I., SEIDEL, H.-P., AND LENSCH, H. P. A. 2008. Fluorescent immersion range scanning. In *Proc. of ACM SIGGRAPH 2008*.

IHRKE, I., GOLDLUECKE, B., AND MAGNOR, M. 2005. Reconstructing the geometry of flowing water. In *ICCV '05*, IEEE Computer Society, Washington, DC, USA, 1055–1060.

KANADE, T., RANDER, P., VEDULA, S., AND SAITO, H. 1999. Virtualized reality: Digitizing a 3d time-varying event as is and in real time. In *Mixed Reality, Merging Real and Virtual Worlds*. 41–57.

MCNAMARA, A., TREUILLE, A., POPOVIC, Z., AND STAM, J. 2004. Fluid control using the adjoint method. In *Proc. of ACM SIGGRAPH 2004*.

MITRA, N. J., FLORY, S., OVSJANIKOV, M., GELFAND, N., GUIBAS, L., AND POTTMANN, H. 2007. Dynamic geometry registration. In *Eurographics Symposium on Geometry Processing*.

MORRIS, N. J. W., AND KUTULAKOS, K. N. 2005. Dynamic refraction stereo. In *Proc. of International Conference on Computer Vision*.

MORRIS, N. J. W., AND KUTULAKOS, K. N. 2007. Reconstructing the surface of inhomogeneous transparent scenes by scatter trace photography. In *Proc. of 11th Int. Conf. Computer Vision*.

QUAN, L., TAN, P., ZENG, G., YUAN, L., WANG, J., AND KANG, S. B. 2006. Image-based plant modeling. In *Proc. of ACM SIGGRAPH 2006*.

SCHNEIDER, R., AND KOBBELT, L. 2001. Geometric fairing of irregular meshes for freeform surface design. *Computer aided geometric design 18*, 359–379.

SHARF, A., ALCANTARA, D. A., LEWINER, T., GREIF, C., AND SHEFFER, A. 2008. Space-time surface reconstruction using incompressible flow. In *Proc. of ACM SIGGRAPH Asia 2008*, vol. 27, 1–10.

SHI, J., AND TOMASI, C. 1994. Good features to track. In *Proc. of CVPR 1994*, 593–600.

SIMON, S. V., BAKER, S., SEITZ, S., AND KANADE, T. 2000. Shape and motion carving in 6d. In *Computer Vision and Pattern Recognition*.

SINHA, S. N., STEEDLY, D., SZELISKI, R., AGRAWALA, M., AND POLLEFEYS, M. 2008. Interactive 3d architectural modeling from unordered photo collections. *Proc. of SIGGRAPH Asia 2008 27*.

STAM, J. 1999. Stable fluids. In *Proc. of ACM SIGGRAPH '99*, 121–128.

STANIFORTH, A., AND CÔTÉ, J. 1991. Semi-lagrangian integration schemes for atmospheric models. *Monthly Weather Review 119*, 9, 2206.

SUN, J., ZHENG, N.-N., AND SHUM, H.-Y. 2003. Stereo matching using belief propagation. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 7, 787.

TAN, P., ZENG, G., WANG, J., KANG, S. B., AND QUAN, L. 2007. Image-based tree modeling. In *Proc. of ACM SIGGRAPH 2007*.

WAND, M., JENKE, P., HUANG, Q., BOKELOH, M., GUIBAS, L., AND SCHILLING, A. 2007. Reconstruction of deforming geometry from time-varying point clouds. In *Eurographics Symposium on Geometry Processing*.

WEI, Y., OFEK, E., QUAN, L., AND SHUM, H.-Y. 2005. Modeling hair from multiple views. In *Proc. of ACM SIGGRAPH 2005*.

WHITE, R., CRANE, K., AND FORSYTH, D. 2007. Capturing and animating occluded cloth. In *Proc. of ACM SIGGRAPH 2007*.

XIAO, J., FANG, T., TAN, P., ZHAO, P., AND QUAN, L. 2008. Image-based facade modeling. *Proc. of SIGGRAPH Asia 2008 27*.

YANG, Q., YANG, R., DAVIS, J., AND NISTER, D. 2007. Spatial-depth super resolution for range images. In *Proc. of CVPR 2007*, vol. 0, 1–8.

ZITNICK, C. L., KANG, S. B., UYTTENDAELE, M., WINDER, S., AND SZELISKI, R. 2004. High-quality video view interpolation using a layered representation. In *Proc. of ACM SIGGRAPH '04*, 600–608.

ZITNICK, C. L., KANG, S. B., UYTTENDAELE, M., WINDER, S., AND SZELISKI, R. 2004. High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics, 23*, 3, 600–608.