Interpolations of Smoke and Liquid Simulations

NILS THUEREY
Technical University of Munich

We present a novel method to interpolate smoke and liquid simulations in order to perform data-driven fluid simulations. Our approach calculates a dense space-time deformation using grid-based signed-distance functions of the inputs.

A key advantage of this implicit Eulerian representation is that it allows us to use powerful techniques from the optical flow area. We employ a five-dimensional optical flow solve. In combination with a projection algorithm, and residual iterations, we achieve a robust matching of the inputs. Once the match is computed, arbitrary in-between variants can be created very efficiently. To concatenate multiple long-range deformations, we propose a novel alignment technique.

Our approach has numerous advantages, including automatic matches without user input, volumetric deformations that can be applied to details around the surface, and the inherent handling of topology changes. As a result, we can interpolate swirling smoke clouds, and splashing liquid simulations. We can even match and interpolate phenomena with fundamentally different physics: a drop of liquid, and a blob of heavy smoke.

CCS Concepts: • **Computing methodologies → Physical simulation**; *Procedural animation*

Additional Key Words and Phrases: Data-driven fluid simulation, interpolation, 5D optical flow

ACM Reference Format:

Nils Thuerey. 2016. Interpolations of smoke and liquid simulations. ACM Trans. Graph. 36, 1, Article 3 (September 2016), 16 pages.
DOI: http://dx.doi.org/10.1145/2956233

1. INTRODUCTION

Fluid simulations are established components of VFX production pipelines, with a variety of powerful solvers to choose from, including Eulerian methods [Stam 1999], hybrids [Zhu and Bridson 2005], and pure Lagrangian approaches [Müller et al. 2003]. Surprisingly, the tools for working with the large amounts of simulation data produced by these solvers are extremely limited—the simulation data is typically just passed on to a rendering stage. Any required change means restarting a new simulation from scratch.

This work was partially funded by the ERC Starting Grant 637014.
Author's address: N. Thuerey, Technical University of Munich, I15; Boltzmannstr. 3; 85748 Garching; email: nils.thuerey@tum.de.

With this work, we target the reuse of simulation data for the automated generation of in-betweens based on a *space-time deformation*. We precompute a matching of two four-dimensional (4D) shapes. Once it is computed, a user can freely choose any version in between the two extremes, which can then be generated very efficiently, without starting a new simulation.

Beyond special effects, this is also highly interesting for interactive, data-driven simulations. Previous work in this area has demonstrated the feasibility of precomputing state graphs that are suitable for games [Stanton et al. 2014]. However, the graph quickly grows in size as all possible interactions have to be explicitly precomputed. In such a setting, our method could be used to interpolate a few key variants, greatly reducing the state graph.

A key challenge for our work is to fully automate the matching process. In contrast to previous work on blending triangle meshes from liquids with user guidance [Raveendran et al. 2014], we target automatic matches of both smoke and liquid effects. The only requirement for our approach is that a signed-distance function on a regular grid is available. The Eulerian representation is especially useful for ensuring spatial and temporal smoothness, which in practice translates into robustness. This makes it possible to directly work with the complex surfaces of splashy liquids, which could otherwise cause misaligned deformations and incomplete matches.

A central insight of this article is the fact that Signed-Distance Functions (SDFs), which are readily available in many flow simulations, are a particularly well-suited input for optical flow. With arbitrary data, the optical flow solve is strongly underdetermined, and will often yield suboptimal or unexpected motions. However, the smooth gradients of SDFs give good results even for large deformations. Relying on the signed-distance property of the inputs also allows us to apply a novel projection step to efficiently recover detailed correspondences. In combination with an iterated residual solving approach that we outline in the following, our approach can find matches between significantly deforming flow surfaces without any user input. Additionally, the volumetric nature and robustness of our approach allow us to calculate matches for the complex shapes of swirling smoke clouds, or even completely different phenomena with changing viscosities.

Specifically, our contributions are

—a novel optical flow approach to register deforming space-time fluid surfaces given by SDFs;
—a method to align multiple consecutive deformations into a single deformation field;
—a multidimensional interpolation scheme for SDFs; and
—an efficient projection to recover small details, in combination with residual iterations and a robust volumetric error measurement.

Together, these contributions lead to a practical algorithm for the robust registration and blending of complex volumetric phenomena.

2. RELATED WORK

Fluid simulations in computer graphics were pioneered by the works introducing stable and efficient grid-based solvers [Foster

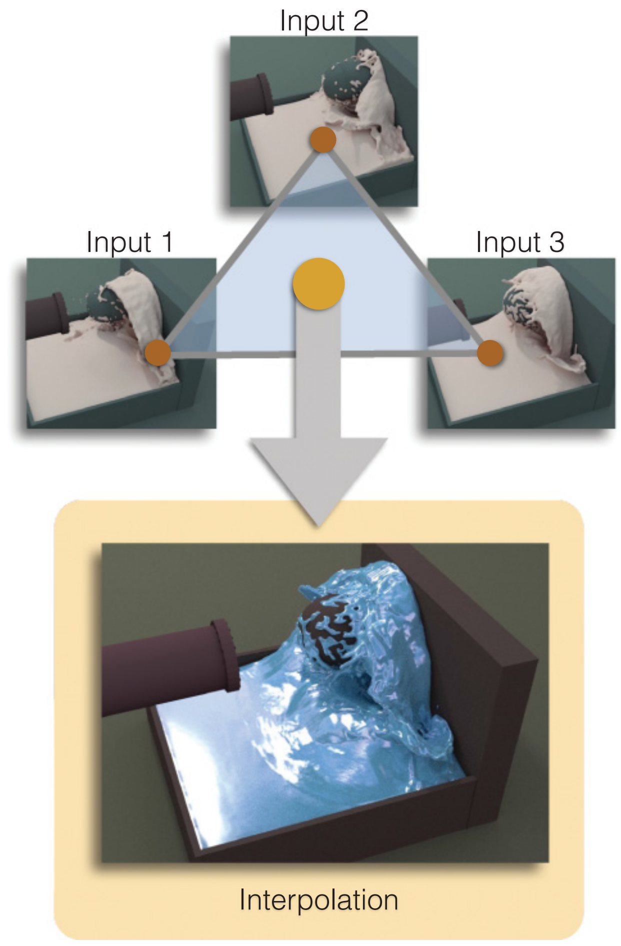

Fig. 1. Our method calculates matches between 4D datasets of smoke and liquid simulations, which are then used to calculate arbitrary in-betweens. A liquid example with a 2D parameter space is shown above.

and Metaxas 1996; Stam 1999], and have since made significant progress. The book by Bridson [2016] gives an excellent overview. We restrict our discussion to Eulerian methods in the following, but powerful Lagrangian techniques are available [Ihmsen et al. 2014]. Our method could work with inputs from arbitrary solvers, as long as they can be converted into signed-distance functions. For liquids, we use inputs from the *Fluid Implicit Particle* (FLIP) approach [Zhu and Bridson 2005], which combines particles and grid data. We also heavily make use of a semi-Lagrangian method [Stam 1999], for which numerous extensions and improvements have been proposed, for example, to increase accuracy with a correction step [Selle et al. 2008], or to add small-scale turbulent detail [Kim et al. 2008].

The aim of our method is closely related to fluid *guiding* approaches, whose goal it is to influence the outcome of a simulation with respect to external, and often nonphysical goals. While early works in this area have mostly focused on guiding shapes [Shi and Yu 2005; Thuerey et al. 2006], recent techniques have introduced more subtle techniques [Pan et al. 2013; Nielsen et al. 2013] that are highly relevant for practical applications. Overall, the aim of these methods differs from our approach: they typically take a single goal surface as input, and then refine or modify the result of a new simulation with respect to this input. Our goal, on the other hand, is to directly interpolate two or more inputs without running new simulations.

Our method can also be seen as a way to precompute special reduced bases for flows. In contrast to previous work in this area [Treuille et al. 2006; Kim and Delaney 2013], our aim is not to capture significant flow motions to build a velocity basis, but to precompute correspondences between the visible shapes of simulations. Thus, our method requires surfaces instead of motions as input. The method of Stanton et al. [2014] is closer to our approach, and captures complex liquid flows by precomputing a state graph that is adapted to player behavior. However, their algorithm does not perform any interpolation on the precomputed data. As such, our contributions are orthogonal, and could help to further reduce the state space in interactive settings. The work by Ladicky et al. [2015] shares our goal to perform data-driven fluid simulations. While we focus on interpolations of simulation data, they represent a broad class of particle interactions for Smoothed-Particle Hydrodynamics simulations with a regression forest.

The work of Stam and Schmidt [2011] explores different possibilities to calculate the surface velocity of a series of implicit surfaces. While we share the goal to reconstruct motions from implicit representations, our approach matches two or more space-time surfaces with dense 4D deformations, instead of reconstructing normal and tangential surface motions of a series of 3D inputs.

Our algorithm employs optical flow, which is a widely used approach to retrieve motions from image data, and we will review several works that are most relevant to our approach from this large field of research here. The seminal work of Horn and Schunck [1981] has been investigated and extended in numerous ways. For our approach we use a hierarchical solve [Meinhardt-Llopis et al. 2013], and employ established best practices from the area [Sun et al. 2014]. A good general overview can be found in books such as the one by Wedel and Cremers [2011]. Early on, Vedula et al. already proposed techniques to reconstruct dense 3D flows [Vedula et al. 1999], in their case based on multiple video streams. Paragios et al. [2003] used optical flow for nonrigid registrations, to estimate a global rigid motion in combination with a local deformation from image data. Optical flow has also very recently been used in combination with fluid simulations to reconstruct velocities from tomographic density data of real flows [Gregson et al. 2014].

While 2D and 3D variants are common, higher-dimensional optical flow solves are rare. One area where 4D solves have been successfully used is registration of data, such as CT scans, in both space and time [Ehrhardt et al. 2007]. In computer graphics, similar techniques have been proposed to reconstruct captured performances with volume conservation [Sharf et al. 2008], and to perform reconstructions in the presence of changing topologies and inconsistencies [Popa et al. 2010]. Space-time data has also been useful for capturing complex phenomena such as trees [Li et al. 2013] and hair [Xu et al. 2014]. However, to the best of our knowledge, we are the first to use optical flow to match multiple space-time datasets, thus effectively performing a 5D solve. All of the preceding methods typically perform a space-time optimization on a single set of 4D data.

The work most similar to ours in spirit is the one by Raveendran et al. [2014]. They noticed that it is crucial to take into account both space and time, to give the optimization enough freedom to find a good match. While they share our goal of computing a space-time registration of surfaces, there are several important differences. First, we target Eulerian SDF surfaces, which are more widely used for surface tracking than meshes. Second, our grid-based representation allows for very efficient regularization, and thus more robust matching. Also, no helper data structures are needed for closest

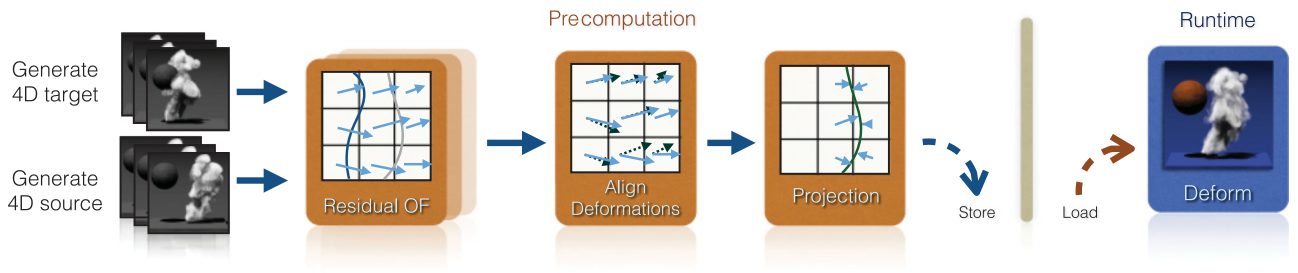

Fig. 2. This figure shows an overview of our interpolation pipeline: the input data is assembled into 4D signed-distance functions, which are analyzed with a 5D Optical Flow (OF) solve. Our deformation alignment procedure combines the deformations from each pass into a single deformation field. Our surface projection step then recovers small-scale details of the target surface, and the final deformations are stored. All that is necessary to generate a new simulation is to apply the precomputed deformations (with a second alignment step for multidimensional interpolations).

point lookups (we can simply follow the gradient of the SDF), the implicit representation inherently handles topology changes, and many operations such as intersections with time planes are trivial and very efficient in our setting. Note that the mesh-based approach [Raveendran et al. 2014] could be run on triangulations of SDFs. However, in this case no vertex correspondences exist, leading to a degradation of quality. Lastly, while a central component of the mesh-based approach was user guiding, we did not require any additional information from a user. Thus, we can run our algorithm automatically on a large number of inputs.

3. METHOD

Given a sequence of surfaces over time from a simulation run, we concatenate these surfaces as time slices of a 4D volume. For a liquid simulation, we simply use the surface of the liquid phase, and for a smoke simulation we use an isolevel of the density volume (details will be given in Section 7). As the 3D signed-distance values do not contain any information about proximity in time, we calculate 4D signed-distance values for this surface. We use an equidistant discretization for the spatial and temporal dimensions of our 4D data, and all dimensions are weighted equally when computing the magnitudes of 4D vectors.[1] Our algorithm takes *two* of these 4D volumes from different simulation setups as input: Φ_1 and Φ_2. It computes a dense field of four-component deformation vectors $\mathbf{u}$, which maps points of Φ_1 onto Φ_2. This deformation has a clear direction. In the following, we will assume this direction to be from Φ_1 to Φ_2, but it is possible to compute the inverse deformation in a separate step.

In the following, we first review the most important concepts of optical flow, before introducing our extensions.

3.1 Optical Flow

The goal of the optical flow step is to compute the deformation $\mathbf{u}$ to transform one input into the other one. The whole nonlinear, and potentially long-range, deformation is decomposed into several smaller, linearized steps, the first few of which we solve with optical flow. For brevity, we will only give a brief overview of the derivation, and then focus on the discrete version of the optical flow solve.

[1] $|(1,0,0,0)^T| = |(0,0,0,1)^T|$. Thus, at $\mathbf{p} = (x, y, z, t)^T$, cell neighbors in space and time have equal distance: $|\mathbf{p} - (1,0,0,0)^T| = |\mathbf{p} - (0,0,0,1)^T|$. We consider inviscid flows; when solving with viscosity, this could be used to relate space and time across simulations.

Notation: For a continuous value (e.g., Φ) we denote its discrete counterpart with a bold symbol (such as $\boldsymbol{\Phi}$). For matrices we will use bold, uppercase letters (e.g., $\mathbf{A}$). We will use one or more $'$ to denote intermediate results. Thus, $'$ will never indicate a derivative.

We make use of a hierarchical variant of the commonly used Horn-Schunck [1981] algorithm. The algorithm is motivated by the so-called *brightness-constancy assumption*. That is, values in an image (or voxels in a volume) move around, but do not change in magnitude. Usually, optical flow considers two images taken at different times. We instead use optical flow to compute correspondence between the space-time surfaces of different simulations. We use a parameter r to reflect the change of the simulation inputs. r could potentially be any parameter changing the outcome of a simulation, for example, the initial position of a drop, viscosity, or even cell size. Given an input $\Phi(x(r), r)$ that moves in space-time with respect to r, the brightness-constancy assumption can be expressed as $\mathrm{d}\Phi/\mathrm{d}r = 0$. Thus, we require Φ to be constant as we change r.

Applying the chain rule yields

$$\frac{\partial \Phi}{\partial r} + \frac{\partial \Phi}{\partial x} \cdot \frac{\mathrm{d}x}{\mathrm{d}r} = 0. \quad (1)$$

This is directly in line with the commonly used material derivative for advection in fluid solvers. The main difference here is that we want to recover the motion of Φ, while fluid solvers typically compute this motion, and aim for computing its effect on Φ. In our setting, the change of position $\frac{\mathrm{d}x}{\mathrm{d}r}$ corresponds to the deformation u.

For further details of the derivation, we refer interested readers to Section 4 and Appendix A of the original Horn and Schunck paper [1981], or books on optical flow, such as the one by Wedel et al. [2011].

The second term of Equation (1) represents a nonlinearity that turns out to be difficult to linearize. Typical fluid solvers take care to compute these terms as accurately as possible with specialized algorithms [Selle et al. 2008], and the nonlinearity is similarly challenging for inverse problems such as optical flow. Next, we will discuss several steps to robustly retrieve solutions.

As solving for brightness constancy alone is strongly underdetermined, a variety of regularizers have been proposed. We employ the two most common regularizers: one penalizing nonsmooth solutions, and a second one to favor deformation vectors with small magnitudes (a so-called *Tikhonov* regularizer). The resulting problem is now formulated as an energy minimization: the goal is to compute a deformation field $\mathbf{u}$, which minimizes the data term

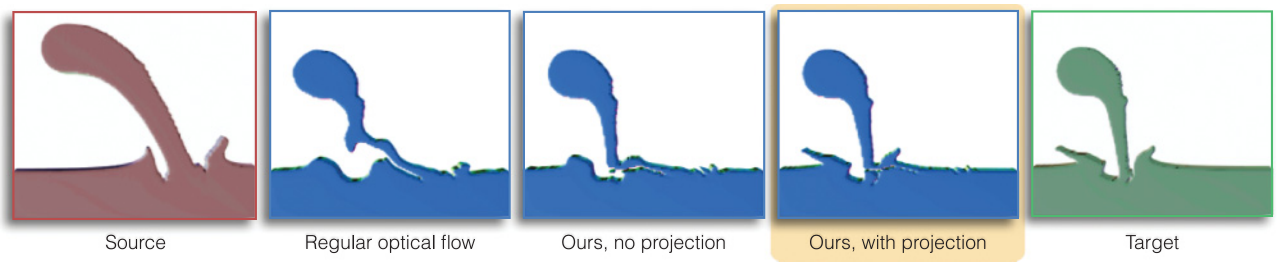

Fig. 3. A 2D example illustrating the components of our pipeline. The images on the far left and right are the source and target inputs. The three center images from left to right are the deformed surface after a single optical flow step, the result after the residual iterations, and one including our projection step. While a single optical flow step does a mediocre job at registering the inputs, our final result closely matches the target surface.

Equation (1), and the two regularizers. We calculate the deformation by minimizing a weighted sum of the energy terms

$$\min_u E_{\text{data}}(u) + \beta_S E_{\text{smooth}}(u) + \beta_T E_{\text{Tikhonov}}(u). \tag{2}$$

The discrete version of the energy to be minimized in a least-squares sense is given by

$$E_{of}(u) = \frac{1}{2}\int_\Omega |\Phi_r + u \cdot \nabla\Phi|^2 + \beta_S \sum_j |\nabla u_j|^2 + \beta_T |u|^2 d\Omega, \tag{3}$$

where β_S and β_T represent scaling factors to control the relative importance of the three terms, and $\sum_j$ sums over the four dimensions. While Φ_r usually denotes the time derivative for regular optical flow applications, here it corresponds to a *data derivative*: the change induced by the aforementioned parameter r, which could be any external control knob modifying the simulation (e.g., a moved inflow position). Thus, our formulation effectively considers five dimensions. We compute the preceding spatial gradient $\nabla\Phi$ with Φ_2.

The discrete minimization of Equation (3) yields a system of linear equations

$$\mathbf{A}_{of}\ \mathbf{u} = \mathbf{b}, \tag{4}$$

which we solve for $\mathbf{u}$. The matrix $\mathbf{A}_{of}$ contains the discretized energy terms, and is given by

$$\mathbf{A}_{of} = [\nabla\mathbf{\Phi_2}]^T[\nabla\mathbf{\Phi_2}] + \beta_S \sum_j \mathbf{L}_j + \beta_T \mathbf{I}. \tag{5}$$

The terms, from left to right, correspond to the aforementioned data term, smoothness, and Tikhonov terms. $[\nabla\mathbf{\Phi_2}]$ denotes an $n \times 4n$ matrix containing the discrete gradient of $\mathbf{\Phi}_2$, and $\mathbf{L}$ denotes the discrete Laplacian. The right-hand side of the linear system is $\mathbf{b} = -[\nabla\mathbf{\Phi_2}]^T \mathbf{\Phi}_r$, where a finite difference with a normalized step size is used to compute $\mathbf{\Phi}_r = \mathbf{\Phi}_2 - \mathbf{\Phi}_1$. The gradients in $\mathbf{A}_{of}$ and $\mathbf{b}$ are 4D vectors along spatial as well as time dimensions.

An inherent difficulty here is to linearize the nonlinear terms of Equation (1). As is common practice for optical flow methods, we use first-order approximations for the corresponding derivatives [Wedel and Cremers 2011]. As a consequence, this basic optical flow solve works well for small deformations (on the order of one to two grid cells), but fails to recover larger motions. We will later on propose an iterative scheme, which we combine with the commonly used hierarchical procedure to recursively resample and deform the data on coarser resolutions [Meinhardt-Llopis et al. 2013]. Both methods help to reduce the inaccuracies of the first-order approximations. The hierarchical scheme does this by solving the equations on different spatial scales, while our iterations combine the results of multiple hierarchical solves.

A central insight of our work is that using SDFs as input for the optical flow solve is paramount for retrieving high-quality deformations. For smoke volumes it might seem natural to calculate optical flow directly on the density data, as we typically perceive smoke clouds to be highly textured. Likewise, one might try to use fill-fraction values for liquid simulations. However, both cases will yield clearly suboptimal results. This problem is the well-known *aperture problem*, which arises from the brightness-constancy assumption: in regions with uniform intensities the motion is completely ambiguous—any point could move to any other one. A typical simulated smoke cloud has zero values outside, and most likely saturated values inside. These regions contain zero information about the motion of the underlying fluid. In typical optical flow applications, this problem is alleviated with regularization, however, it is preferable to minimize the ambiguity in the first place. This is where SDFs are highly beneficial: by definition, they have values that change with distance to the implicitly defined surface at the zero level set. While there is still ambiguity with respect to the tangential direction of the surface, a band of values normal to the surface can now be robustly matched.

In the optical flow solve, the motion is expressed in terms of the gradient of the input data. Suboptimal input data, with saturated or empty regions, has insufficient, and ambiguous gradient information as a consequence, and spatial gradients exist only in a relatively small band. The sparsity of such data is illustrated for a smoke cloud in Figure 4. The center image shows gradient magnitudes, and the predominantly black areas of the image indicate a complete lack of gradient information. In contrast, an SDF has smooth and reliable gradients even far away from the surface (effectively, as far as distance information was generated). Figure 4 (right) shows gradient magnitudes for an SDF generated for the 0.5 level set of the smoke cloud on the left. The uniform bright blue color indicates gradients of length one. The original surface is not shown. It lies right in the middle of the thick blue band. The gradients could easily fill the whole image. The drop off to zero only happens because the SDF was truncated at a distance of 15 cells in this example. The medial axis of the SDF is also visible as black lines in the interior of the shape.

The gradients of the input data make up the core diagonal blocks of matrix A_{of} of Equation (4). As such, they are crucial for a well-posed optical flow problem. The matrices from SDF data typically have much more reliable deformations, as the gradient terms on the diagonal guide the solution in a broad band around the surface. As additional benefit, using 4D SDFs allows us to match small and fast moving structures: even fast moving parts of the surface will have a large *halo* of 4D distance values, on which the optical flow can operate.

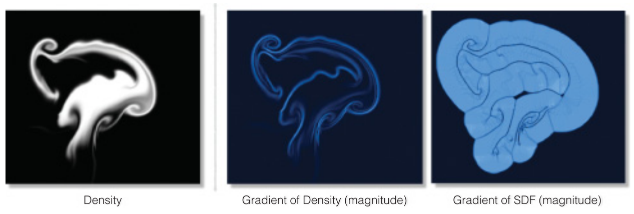

Fig. 4. These images highlight the importance of using an SDF for the optical flow solve. An example input density is shown on the left, while the next two images show magnitudes of gradients in blue. Larger gradients are shown with brighter shades of blue. The center image was generated from gradients calculated based on the input density, while the right image was calculated from an SDF of the 0.5 level set of the input. As these gradients make up the block-diagonal band of the optical flow matrix $\mathbf{A}_{of}$, reliable gradients are crucial, and the area of bright blue values in the middle and right images directly indicates how many reliable gradients can be used to assemble the matrix in each case.

3.2 Residual Optical Flow

If a first deformation $\mathbf{u}^1$ calculated by optical flow brought us closer to the target, we can apply the current deformation, and perform another solve for the remaining difference, that is, the residual, yielding a second deformation $\mathbf{u}^2$. This works particularly well in a setting where we have a clear target without noise or other measurement errors. Here this goal state is exactly the target SDF $\mathbf{\Phi}_2$. We can continue to refine the deformation with residual iterations until either a computational budget is exhausted, or the refinement does not yield gains in quality anymore. To combine the sequence of deformation fields $\mathbf{u}^{1..n}$ from each solve into a single one we employ the *deformation alignment* from Section 4. The difficulties that arise when combining multiple Eulerian deformations, as well as our solution will be explained in full detail there. As a result, we retrieve a single deformation that aligns the two surfaces significantly better than only one optical flow solve (see Figure 3, middle).

Conceptually, these residual iterations are important, as the optical flow solve performs the aforementioned linearization of Equation (1) around the input states. The iterations relinearize the problem closer and closer to the target state, and in combination with the hierarchical solve can significantly improve the final quality of the deformation. Previous works have considered directly solving the nonlinear problem with Newton's method [Zikic et al. 2010], but generic nonlinear solvers are typically outperformed by methods that employ specialized algorithms for advection (e.g., in combination with the hierarchical scheme that we employ [Meinhardt-Llopis et al. 2013]). Other variants of optical flow solvers re-formulate the equations by splitting of the deformation into a fixed part and an incremental deformation. The increment is retrieved by repeated optical flow solves [Zach et al. 2007], typically also in combination with a prewarping step that uses an advection algorithm.

In contrast, our approach is based on aligning the different deformation fields with the method from Section 4. While our method yields results that are similar to the incremental variants, our method has the advantage that additional deformation fields could be incorporated by alignment. For example, future extensions of our algorithm could improve the matching of two inputs by postprocessing the deformation computed by optical flow. Our alignment could then be used to combine the optical flow deformation, and one or more postprocessing deformations into a single one. An additional smaller advantage is that given a function for aligning deformations, the residual iterations can be easily implemented. Thus, this approach seamlessly integrates into our interpolation framework.

3.3 Surface Projection

The optical flow step does a good job at robustly detecting smooth large-scale motions, but its inherent regularization prevents it from matching fine details at the surface. To retrieve this detail, we can leverage knowledge about the input data: it is a signed-distance function, and we know that the source surface should ideally deform to end up exactly on the target surface. If the surfaces are sufficiently close, such a deformation can be easily obtained by marching along the negative gradient of the SDF. This projection is inspired by techniques from mesh-based registration [Bojsen-Hansen et al. 2012]. We will show that it can be adapted to a volumetric setting, and leads to considerable gains in quality regarding small-scale detail.

Given an SDF deformed by a deformation from the optical flow step, we perform a bisection search for each cell along the gradient of the target SDF, until we find the spot on the target where it matches the isosurface value of the source cell. This line search has the advantage of being trivially parallel, as the result depends purely on the deformed SDF and target SDF. We consider the projection as an update step for a current deformation estimate $\mathbf{u}^k$:

$$\mathbf{u}^{k+1}(\mathbf{x}) = \mathbf{u}^k(\mathbf{x}) + \text{bisect}\big(\mathbf{x} + \mathbf{u}^k(\mathbf{x}), -\mathbf{n}(\mathbf{x} + \mathbf{u}^k(\mathbf{x}))\big), \qquad (6)$$

where the function *bisect* performs the actual search from the start position (first parameter) along the direction passed as the second parameter. $\mathbf{n}$ in this case denotes the normalized gradient of the target $\mathbf{\Phi}_2$ and is inverted for points inside the volume.

The completely independent calculation of the projection for each cell can yield different deformations in regions of quickly changing target normals, and the calculation of the direction is unreliable on the medial axis of the SDF. However, we found that the projection step can yield very good results if it is restricted to a narrow band of width τ_{proj} around the surface, and its output is smoothed using a Gaussian of size σ_{proj}. To propagate the effect of the projection into the volume around the surface, we extrapolate the deformation values before applying the smoothing. We do this with τ_{proj} explicit iterations [Jeong and Whitaker 2008], fading out the deformation over the course of the iterations. As illustrated in Figure 3 the combined deformation snaps the input very tightly to the target surface, recovering small-scale details in this way.

Note that this projection step solves essentially the same problem as the optical flow step. However, due to the aforementioned difficulties, it is no replacement for it. Instead, the projection only gives good results once a suitable overall match has been obtained by optical flow.

3.4 Error Metric

A last component that is important for combining the aforementioned techniques is a reliable metric to quantitatively evaluate a deformation. Unlike settings where optical flow is traditionally employed (e.g., to calculate motions in video data), our synthetic data has no occlusions, or motion blur, which motivates our choice of metric. Alternatively, the energy from Equation (3) could be used to evaluate the final quality, but we found that discarding the regularization terms, and putting an emphasis on the volume itself yields results that are more in line with the perceived alignment of the surfaces. While an algorithm to directly minimize the following metric is imaginable, we leave this for future work, and purely use the error metric as a stopping criterion for our optical flow and projection iterations.

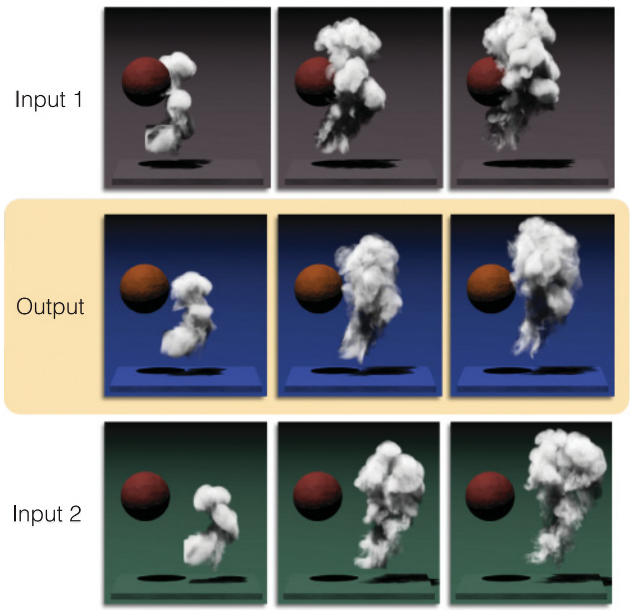

Fig. 5. An example deformation for a smoke simulation. The inputs on the top and bottom are matched with *FlOF* and interpolated for a weight of 0.5 in the middle. Our interpolation recovers the translation along x, as well as the deformation of the cloud.

We define the error metric as the integral of an indicator function h over the domain. In a discrete setting given two input SDFs $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$ (which we assume to be calculated for a normalized cell size of 1), this becomes

$$e(\mathbf{\Phi}_1, \mathbf{\Phi}_2) = \sum_{\mathbf{x}} h(\mathbf{\Phi}_1(\mathbf{x}), \mathbf{\Phi}_2(\mathbf{x}))V, \qquad (7)$$

where V is the volume of a cell. The question of which function to choose for h has been explored in various settings, for example, for mesh similarity [Cignoni et al. 1996], or within optimizations for shape registration [Cremers and Soatto 2003]. We have found that a metric detecting nonmatching volumes is especially important in our setting:

$$h(s_1, s_2) = \begin{cases} 0 & \text{if } \operatorname{sgn}(s_1) = \operatorname{sgn}(s_2) \\ \min(1, |s_1 - s_2|/\Delta x) & \text{otherwise.} \end{cases} \qquad (8)$$

This metric has a value of at most 1 where the two SDFs disagree, and is zero where the SDFs agree, but still detects subcell shifts.

We have tried alternate metrics such as a simple difference of the two SDFs ($h = |\mathbf{\Phi}_1 - \mathbf{\Phi}_2|$), the *metro* distance mentioned previously [Cignoni et al. 1996], or the energy of the optical flow solve, but we found that our version from Equation (8) is more reliable in practice. What turned out to be most important is a metric that puts more emphasis on the surface region. When using error metrics based on direct differences of the two inputs, large distance values easily led to undesirably large error values. Thus, the other metrics tend to introduce large errors when small pieces of the surfaces disagree (e.g., drops far from the bulk volume). However, this is visually not as crucial as a good match of the large-scale volumes of the liquid surface, which is why our metric puts more emphasis on the latter.

The comparison in Figure 6 demonstrates that our metric is more reliable for detecting differences than the optical flow energy itself. In both cases, a deformed surface is shown in yellow, while the surface of the target is shown in dark blue in the background. The optical flow solves were performed with high accuracy, and have a very small residual of 10^{-15} for both cases. Visually, a difference is clearly noticeable, but the averaged optical flow energy cannot detect the remaining differences. Instead, our error metric detects a residual difference for both versions, and yields a three times higher value for the right input. To purely illustrate the effect of our error metric versus the optical flow energy, we have disabled the surface projection and residual iterations for this example.

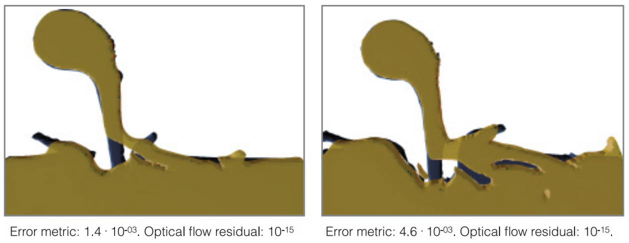
Error metric: 1.4 · 10-03. Optical flow residual: 10-15
Error metric: 4.6 · 10-03. Optical flow residual: 10-15.

Fig. 6. This image illustrates the advantages of our error metric for two different timesteps of a two-dimensional input: while both final deformed surfaces (in yellow) have very similar optical flow residuals close to zero, our error metric detects a significant difference. The higher error is caused by the broken up surface in the center region of the right image. The target surface is shown in dark blue in the background.

ALGORITHM 1: Pseudocode for our full algorithm. Parameters are given in Table I.

```
function FlOF($\mathbf{\Phi}_1$, $\mathbf{\Phi}_2$, $\mathbf{u}$)
if $gridsize(\mathbf{\Phi}_1) \geq s_{max}$ then
    $\mathbf{\Phi}'_1, \mathbf{\Phi}'_2, \mathbf{u}'$ = Downsample($\mathbf{\Phi}_1, \mathbf{\Phi}_2, \mathbf{u}$)
    $\mathbf{u}'$ = FlOF($\mathbf{\Phi}'_1, \mathbf{\Phi}'_2, \mathbf{u}'$) // recurse
    $\mathbf{u} = \mathbf{u}$ + Upsample($\mathbf{u}'$)
end
for $l = 1$ to $l_{max}$ do
    $\mathbf{\Phi}''_1$ = advect($\mathbf{\Phi}_1, \mathbf{u}, 1$)
    $\mathbf{u}_l$ = opticalFlow( $\mathbf{\Phi}''_1, \mathbf{\Phi}_2, \sigma_{\text{of}}$ )
    $\mathbf{u}_{\text{tmp}} = \mathbf{u}$ + alignVelocity($\mathbf{u}_l$)  // See Section 4
    if $e(advect(\mathbf{\Phi}_1, \mathbf{u}_{tmp}, 1), \mathbf{\Phi}_2) \leq e(advect(\mathbf{\Phi}_1, \mathbf{u}, 1), \mathbf{\Phi}_2)$ then
        $\mathbf{u} = \mathbf{u}_{\text{tmp}}$
        $\sigma_{\text{of}} = 3/4\,\sigma_{\text{of}}$
    end
    else
        break
    end
end
if is finest grid($\mathbf{\Phi}_1$) then
    for $k = 1$ to $k_{max}$ do
        $\mathbf{u}$ += project($\mathbf{\Phi}_1, \mathbf{\Phi}_2, \mathbf{u}, \sigma_{\text{proj}}$)
        $\sigma_{\text{proj}} = 3/4\,\sigma_{\text{proj}}$
    end
end
return u
```

3.5 Full Algorithm

Algorithm 1 outlines the integration of our residual OF and projection steps into the hierarchical optical flow scheme. Here the solution of the optical flow solve is smoothed with a Gaussian

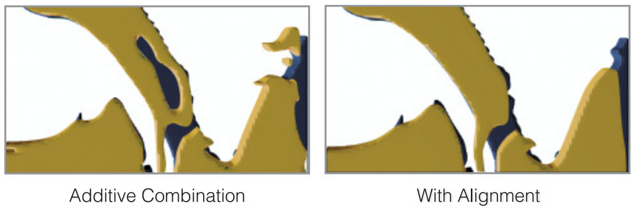

Fig. 7. The effects of our deformation alignment for a practical example: Using an additive combination (left image), structures can break up or artifacts can develop, for example, in the top-right corner of the left image. The target surface is shown in dark blue in the background.

kernel of size σ_{of}, before passing the deformation on to the next higher level (or the caller).

For the final algorithm we adopted another strategy from the mesh-based registration area: it is more robust to first match larger scales, before moving to smaller ones. We can do this within a single level of the hierarchy by varying the blur kernel radius. Thus, when iterating either the optical flow or projection, we start with a relatively large kernel, and then reduce its size by a factor of 3/4 after each step. We use our error metric from Section 3.4 to check whether the last optical flow solve leads to an overall improvement, or whether it is preferable to stop iterating.

The full matching algorithm, which we denote with *FlOF* (for FLuid Optical Flow), is summarized in Algorithm 1. Initially, our algorithm is called with the two input SDFs, and a zero velocity $\mathbf{u} = 0$. The *advect*$(\mathbf{a}, \mathbf{v}, \alpha)$ function applies the deformation $\mathbf{v}$ to $\mathbf{a}$ with weight $\alpha \in [0..1]$.

4. DEFORMATION ALIGNMENT

In the following, we will describe our new approach to align multiple consecutive Eulerian deformations so that they can be merged into a single deformation field. This alignment is useful for the preceding residual iterations, each of which produces a separate deformation. It is also crucial for the higher-dimensional interpolations of Section 5, to align the results of multiple *FlOF* solves. In both cases, it is highly preferable to compute a single, smooth linear deformation from source to target, instead of storing and applying sequences of deformations. The importance of the alignment for a practical 2D deformation example can be seen in Figure 7. We will first describe the alignment for two deformations, before considering arbitrary sequences.

Given an input surface Φ and two deformations u_1, u_2 with scaling factors α_1, α_2, respectively, we assume for now that each deformation is applied with a semi-Lagrangian lookup $\Phi'(x) = \Phi(x - u(x))$. Our approach would likewise work with higher-order methods [Selle et al. 2008]. Here the $'$ indicates that Φ' is a temporary value, which is only required for the next calculation step, and can be discarded afterwards. We arrive at the final configuration Φ_{dst} with

$$\begin{aligned} \Phi'(x) &= \Phi(x - \alpha_1 u_1(x)), \\ \Phi_{dst}(x) &= \Phi'(x - \alpha_2 u_2(x)). \end{aligned} \tag{9}$$

Our goal is to reach Φ_{dst} with a single advection calculation using the unknown deformation $\mathbf{u}_{comb}$: $\Phi_{dst}(x) = \Phi(x - \mathbf{u}_{comb}(x))$.

Combining these deformations is difficult primarily due to the Eulerian representation. Eulerian advection methods are typically *backwards* looking, that is, pull data towards a sample location. Thus, if we consider a surface to be deformed, the deformation vectors acting on the surface are not located at the actual surface

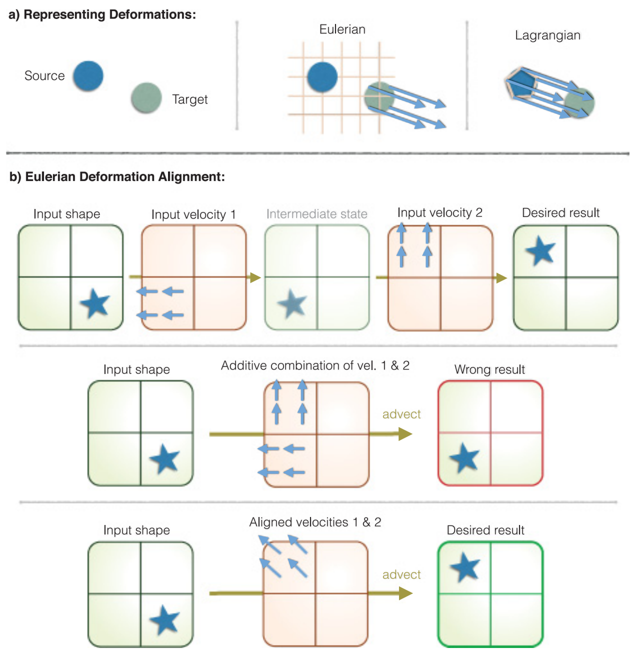

Fig. 8. (a) A sample deformation (left) can be represented in an Eulerian way (middle, only relevant deformation vectors shown) or a Lagrangian way (right). For the latter, the deformation vectors are located at the source position (i.e., at vertices or particles), while the Eulerian setting we are using requires the deformation to be stored in the cells of the target position. (b) An example highlighting the importance of aligning Eulerian deformations: the top row shows the input SDF (a star shape), that is deformed with two consecutive deformations (orange fields). An additive combination of the two deformations effectively ignores the second one, while the result of our alignment (bottom) gives the desired result with a single step.

positions, but instead in the target region where the surface should be moved to. For every spatial position, for example, every cell in a grid, we perform a backwards lookup along the velocity direction to locate which data should be moved to the cell under consideration. This is in contrast to a Lagrangian representation, where the deformation is stored directly at the surface, and each surface point is moved forward with its deformation vector. This difference is illustrated in Figure 8(a). While combining two deformations is trivial in Lagrangian settings—they can simply be added—this approach yields incorrect results for Eulerian representations. Intuitively, the problem is that the deformations for a single surface point are located at two different locations, one for each deformation.

It is also important here that our Eulerian deformations vary spatially. Simple additions would work for cases such as uniform translations, but the deformations we are interested in often move different parts of the surface to different target locations, and as such are typically far from uniform. In the following, we will explain how to align two or more of these deformations, such that they can be merged into a single Eulerian deformation field.

The setup of Figure 8(b) illustrates the problem: we start with an input shape Φ in the bottom-right quadrant, and the two deformations (orange boxes) shown in the top row of Figure 8(b). Directions of the vectors in relevant quadrants are indicated with blue arrows. Each deformation is assumed to move the input surface from one quadrant of the domain to the next one for $\alpha_1 = \alpha_2 = 1$. The deformation vectors in empty quadrants do not matter in this example.

We can assume that they are relatively small, and point in a different direction, for example, to the right. Deformation u_1 moves Φ left, while u_2 moves it upwards, so that the target configuration has Φ positioned in the top-left quadrant. The Eulerian representation of the deformations in this case means that the deformation vectors moving the star to the left in u_1 are typically not present at the actual surface locations of the star, but in the target quadrant. The straightforward combination of u_1 and u_2 by addition yields

$$\Phi(x - (u_1(x) + u_2(x))), \tag{10}$$

which is clearly wrong and completely ignores u_1 (as shown in the middle row of Figure 8(b)).

Instead, it is necessary to align the deformation lookup for u_1 with the second deformation, as the combined deformation needs to have the right deformation vectors in the target region where the surface should end up. Thus, a location x with deformation $u_1(x)$ is not applied to x for the combined deformation, but it is moved by u_2. Instead, the deformation $u_1(x - u_2(x))$ is the one that is applied at x.

Thus, it is necessary to apply the deformation u_2 to u_1. We do this by computing an intermediate, aligned deformation u'_1 with

$$u'_1(x) \;=\; u_1(x - u_2(x)), \tag{11}$$

which can now be combined with u_2 as indicated in Equation (10) with

$$\Phi_{\text{dst}}(x) \;=\; \Phi(x - \alpha_1 u'_1(x) - \alpha_2 u_2(x)). \tag{12}$$

This aligned combination is shown on the bottom row of Figure 8(b). Here, the deformation correctly combines both left and upwards motions in the top-left quadrant, moving the input shape with a single advection step. Note that α_2 is only applied during the addition in Equation (12), and not yet in Equation (11).

This deformation alignment extends to arbitrary sequences. The process is illustrated for the following three deformations. In this case, Φ_{dst} is calculated by

$$\begin{aligned}\Phi'(x) &= \Phi(x - \alpha_1 u_1(x)),\\ \Phi''(x) &= \Phi'(x - \alpha_2 u_2(x)),\\ \Phi_{\text{dst}}(x) &= \Phi''(x - \alpha_3 u_3(x)),\end{aligned} \tag{13}$$

which can be expressed in terms of aligned deformations as

$$\begin{aligned}u'_1(x) &= u_1(x - u_3(x) - u_2(x - u_3(x)))),\\ u'_2(x) &= u_2(x - u_3(x)),\\ \Phi_{\text{dst}}(x) &= \Phi(x - \alpha_1 u'_1(x) - \alpha_2 u'_2(x) - \alpha_3 u_3(x)).\end{aligned} \tag{14}$$

Note that both u_1 and u_2 have to be aligned with u_3 in this case. Thus, while the last deformation of a sequence is left unmodified, previous deformations need to be aligned by all previous aligned deformations. This leads to our final algorithm for combining n deformations:

```
function alignVelocity($\mathbf{u}_1, \dots, \mathbf{u}_n$)
$\mathbf{u}_{\text{comb}} = \alpha_1 \mathbf{u}_1$
for $i$=2 to $n$ do
    // Alignment via semi-Lagrangian step
    $\forall \mathbf{x}: \mathbf{u}_{\text{tmp}}(\mathbf{x}) = \mathbf{u}_{\text{comb}}(\mathbf{x} - \mathbf{u}_i(\mathbf{x}))$
    $\mathbf{u}_{\text{comb}} = \alpha_i \mathbf{u}_i + \mathbf{u}_{\text{tmp}}$
end
return $\mathbf{u}_{comb}$
```

Like before, the scaling factors α_i are only applied when accumulating deformations in $\mathbf{u}_{\text{comb}}$, but not when applying them to the previous deformations.

5. INTERPOLATION

The deformations are typically calculated for a set of inputs during a preprocessing stage. We now explain the runtime interpolation step to generate new outputs with these deformations. Note that we always calculate the deformations for SDF inputs, but we apply those deformations to the original dataset (e.g., either a liquid SDF or a smoke volume) when generating an interpolated version. Thus, the inputs, denoted by $\boldsymbol{\Psi}$ in the following, are not necessarily SDFs.

To perform an interpolation we require a series of input datasets $\boldsymbol{\Psi}_i$ with parameter values $\mathbf{r}_i$. Here, the vector $\mathbf{r}_i$ can represent arbitrary parameters of the input simulations, for example, the xy-position of an inflow object in a plane, or its size. The dimension of $\mathbf{r}_i$ directly determines how many dimensions need to be interpolated to generate an output surface. After describing our general approach in the following, we will give the details of the 1D and 2D versions, and then discuss a modification for liquids.

To interpolate a new output with chosen parameters $\tilde{\mathbf{x}}$ we first connect our input data points with appropriate simplices. Thus, for a single parameter, they are connected with line segments, and for a 2D parameter space by triangles. Higher dimensions correspondingly require tetrahedra or higher-dimensional simplices to discretize the volume of the parameter space.

We aim for an interpolation scheme that yields a smooth transition between the inputs and that retrieves the inputs at its endpoints. To generate an output for the parameters $\tilde{\mathbf{x}}$, we find the simplex that contains $\tilde{\mathbf{x}}$, and transform $\tilde{\mathbf{x}}$ into barycentric coordinates. For this we use a regular barycentric conversion

$$H(\tilde{\mathbf{x}}, \mathbf{r}_1, \dots, \mathbf{r}_n) = \begin{pmatrix}\mathbf{p}'\\ 1 - p'_1 - \dots - p'_{n-1}\end{pmatrix} \text{ with} \tag{15}$$
$$\mathbf{p}' = ((\mathbf{r}_1 - \mathbf{r}_n)\dots(\mathbf{r}_{n-1} - \mathbf{r}_n))^{-1}(\tilde{\mathbf{x}} - \mathbf{r}_n)$$

that maps the point $\mathbf{p}$ onto the barycentric space of the $(n-1)$D simplex S spanned by $\mathbf{r}_1, \dots, \mathbf{r}_n$. We then deform the input datasets and interpolate them with the barycentric coordinates

$$\mathbf{x} = H(\tilde{\mathbf{x}}, \mathbf{r}_1, \dots, \mathbf{r}_n). \tag{16}$$

The deformations need to cover the whole simplex, but we do not require deformations for all possible connections between the vertices of a simplex. Instead, we order the deformations to span all sides, and thus require $d+1$ deformations for a d-dimensional simplex. This is illustrated by the green arrows in Figure 9 for two examples. Each deformation $\mathbf{u}_{i\to j}$ is calculated for the pair of inputs (i, j) with Algorithm 1.

The simplest possible case is a single parameter x_1 and two inputs $\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2$ with deformations $\mathbf{u}_{1\to 2}$ and $\mathbf{u}_{2\to 1}$. In this case, the result $\boldsymbol{\Psi}_o$ is given by

$$\begin{aligned}(x_1, x_2)^T &= H(\tilde{x}, r_1, r_2),\\ \boldsymbol{\Psi}'_1 &= \text{advect}(\boldsymbol{\Psi}_1, \mathbf{u}_{1\to 2}, 1 - x_1),\\ \boldsymbol{\Psi}'_2 &= \text{advect}(\boldsymbol{\Psi}_2, \mathbf{u}_{2\to 1}, 1 - x_2),\\ \boldsymbol{\Psi}_o &= x_1\, \boldsymbol{\Psi}'_1 + x_2\, \boldsymbol{\Psi}'_2.\end{aligned} \tag{17}$$

The inputs are deformed for the desired parameter position, and the deformed intermediate datasets $\boldsymbol{\Psi}'_i$ are then blended with the barycentric weights. Note that the deformations to calculate $\boldsymbol{\Psi}'_1$ and $\boldsymbol{\Psi}'_2$ are applied with a factor of 1 minus the other barycentric weight in Equation (17). It is also worth pointing out that it is typically

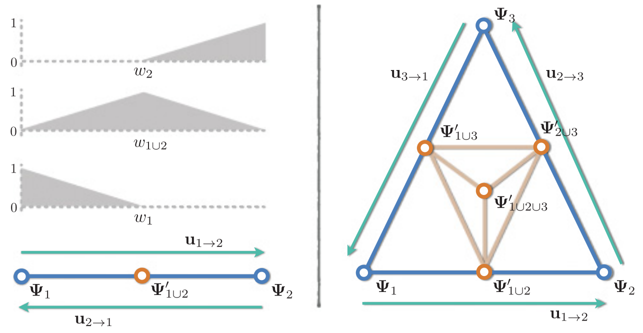

Fig. 9. Our subdivision for simplices spanning the parameter space of liquid input data can be seen for a 1D (bottom-left) and 2D example (right). The orange dots indicate inserted data points from SDF unions. The weights for each of the three data points of the 1D example are shown on the left side.

not necessary to calculate the deformed inputs Ψ' for the whole 4D volume at once. We perform the necessary calculations only for the 3D slice of the inputs that should be displayed next, and then scale and add these slices. This significantly reduces memory requirements for higher dimensions.

This approach directly extends to 2D input parameters, by using barycentric coordinates of triangles spanning three inputs. However, it is necessary here (as well as for higher dimensions), to align the sequence of deformations. As in Section 4, we have consecutive deformations that are applied to an input. A straightforward combination would only work when both deformations contain pure translations. For lower values, an alignment of earlier deformations using our algorithm from Section 4 is crucial to prevent undesirable motions when combining the deformation. Thus, for two dimensions, we apply deformations $\mathbf{u}_{i\to i+1}$ and $\mathbf{u}_{i+1\to i+2}$ to Ψ_i, with weights $(1-x_i)$ and $(1-x_i-x_{i+1})$, respectively, where $\mathbf{u}_{i\to i+1}$ is aligned with Equation (11).

Smoke. For smoke inputs, the deformations calculated by the optical flow step do not necessarily conserve mass, nor do the inputs necessarily have matching total masses over time. For the former, we calculate the total mass after applying the deformation, and normalize it to yield the original mass of the input. This normalization factor is calculated for each smoke deformation separately. The transition from source to target mass is then handled by the linear interpolation.

Liquids. While this linear blend works nicely for smoke data, we found that the union blending technique of Stanton et al. [2014] yields higher-quality results for the SDFs of liquids. They propose to blend via the union of both SDFs, which is especially important for droplets and thin structures. While smaller wisps of smoke simply become transparent when they are scaled down during interpolation, smaller features of SDFs can quickly disappear. Blending via the union of the input SDFs preserves these small structures much better.

We first review the union-blending approach [Stanton et al. 2014] for two inputs in the following, and then extend this idea to higher dimensions. In settings where a high quality is not crucial, for example, for real-time applications, unidirectional *nearest-neighbor* interpolation could be used. For a set of chosen weights, the result is then calculated by deforming only the closest input, without blending other data points. This is in line with the nearest-neighbor interpolation from Raveendran et al. [2014].

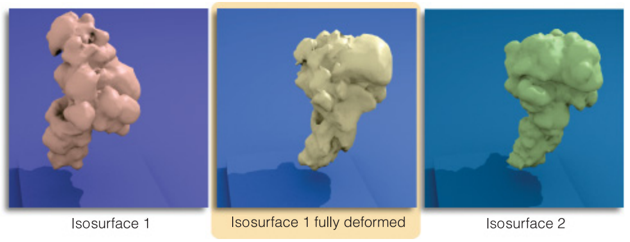

Fig. 10. Isosurfaces extracted from smoke simulations. The source and target are on the left and right, respectively. In the middle the fully deformed source surface is shown. It closely matches the medium to large-scale features of the target. This example uses only a single deformation, and no blending between the two inputs.

For all our results, we use the following approach with union blending. For the simplest case, two inputs Ψ_1 and Ψ_2 from liquid simulations, and weight α, this yields

$$\begin{aligned} \Psi'_{1\cup 2} &= \min(\Psi'_1, \Psi'_2), \\ \Psi_o &= \omega_1\, \Psi'_1 + \omega_{1\cup 2}\, \Psi'_{1\cup 2} + \omega_2\, \Psi'_2, \text{ with} \end{aligned} \tag{18}$$

$$\omega_1 = \text{clamp}(1-2\alpha),\ \omega_2 = \text{clamp}(2\alpha - 1), \tag{19}$$

where clamp ensures a 0 to 1 range, and $\omega_{1\cup 2} = 1 - \omega_1 - \omega_2$. The main differences to before are the different interpolation weights, and the inclusion of the temporary union SDF $\Psi'_{1\cup 2}$ from the deformed inputs. This interpolation requires slightly more memory to store $\Psi'_{1\cup 2}$, but in practice runs as fast as the simpler version from Equation (17).

We now extend our preceding general formulation to include the union SDFs and propose a simple scheme to calculate the corresponding weights for higher dimensions. For this we subdivide each initial simplex S into smaller simplices S_i by adding data points for the union SDFs. One such data point is added at the center of each lower-dimensional simplex. Thus, for a 1D interpolation, the union of the two inputs is added at the midpoint of the line, yielding the interpolation from Equation (18). In 2D we add three union data points along the edges of the triangle, and one union in the center. This subdivision scheme is illustrated in Figure 9 for 1D and 2D, and easily extends to higher dimensions.

We calculate the subdivision and the interpolation weights in terms of the barycentric coordinates of the initial simplex. We then check in which simplex S_i the point $\mathbf{x}$ lies, and determine the weights for the data points involved with a suitable mapping onto the local barycentric coordinates of S_i. All other interpolation weights are set to zero. With this scheme the weight calculation of Equation (18) can be reformulated in the following way:

$$\begin{aligned} \mathbf{x} &= H(\tilde{\mathbf{x}}, r_1, r_2), \\ \begin{pmatrix} \omega_1 \\ \omega_{1\cup 2} \end{pmatrix} &= H(\mathbf{x}, 0, \tfrac{1}{2}) \text{ if } \mathbf{x} \in S(0, \tfrac{1}{2}), \\ \begin{pmatrix} \omega_{1\cup 2} \\ \omega_2 \end{pmatrix} &= H(\mathbf{x}, \tfrac{1}{2}, 1) \text{ if } \mathbf{x} \in S(\tfrac{1}{2}, 1). \end{aligned} \tag{20}$$

The final blending of the two deformed input SDFs and their union is performed as in Equation (18).

This approach naturally extends to 2D parameter spaces. The full set of weight calculations can be found in the Appendix. An exemplary calculation of the weights for a point in the bottom-right

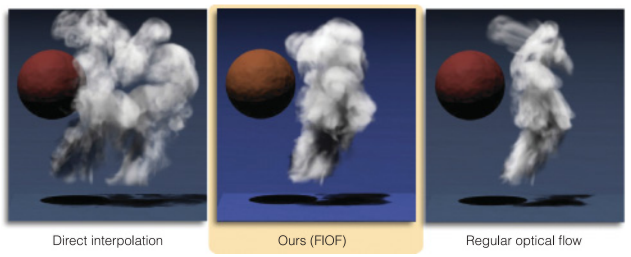

Fig. 11. From left to right: a direct blend of the two input volumes, a bidirectional blend with FlOF, and a bidirectional blend using a regular optical flow match computed from the smoke densities. The direct interpolation gives an undesirable duplication, while the regular optical flow result exhibits a clearly suboptimal match. Our version in the middle leads to a very good alignment of the two inputs.

triangle of Figure 9 (right) is

$$\begin{pmatrix} \omega_{1\cup 2} \\ \omega_2 \\ \omega_{2\cup 3} \end{pmatrix} = H\left(\mathbf{x}, \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ \frac{1}{2} \end{pmatrix}\right). \tag{21}$$

The unions are calculated with a negligible cost from the inputs deformed by two aligned deformations. Our approach can be used to efficiently calculate high-quality interpolations of liquid datasets, and the barycentric calculation of subdivision and interpolation weights simplifies calculations for higher dimensions.

6. IMPLEMENTATION

Using Cartesian Eulerian grids has the advantage that changes of resolution are trivial, which is very useful to reduce the size of the optical flow solve. Equation (4) is linear, and theoretically simple to solve, but the high dimensionality of the datasets can lead to long runtimes. In practice, we downsample the inputs and run the optical flow solve with a resolution of around 60^4. To solve the linear system, we use a conjugate gradient solver, and impose $\mathbf{u} = 0$ at the domain boundary. As the optical flow matrix can be ill-conditioned, we recommend using a diagonal preconditioner. This significantly speeds up runtimes, and performs better than more complex preconditioners in our tests. The deformations can then be up-sampled to generate higher resolution outputs. The 5D optical flow solve itself requires about 5GB of memory with our implementation (for 60^4). To reduce the memory requirements when generating the final interpolations, it is only necessary to keep a window corresponding to the maximal deformation in time in memory. For our examples, we chose a temporal window of 20% of the simulation length, which encompasses the largest encountered time offsets. We also found that the best parameters for our algorithm are not input dependent and we used the set shown in Table I for all our examples. Note that we assume a normalized cell size, thus $\Delta x = 1$, and SDF data that is in the range $[-\gamma_{\max}, \gamma_{\max}]$. These distance values are scaled by β_{image} for the OF solve.

We evaluated a variety of options for each of the algorithmic components of the optical flow. For example, different methods to interpolate, blur, and advect the surfaces are imaginable. We were surprised to find that some of the recommended best practices for optical flow [Sun et al. 2014] did not lead to better deformations. For example, median filtering and cubic interpolation did not reduce the error. Especially the latter led to increased error measurements for some tests. Likewise, substepping the advection for different CFL conditions, or using higher-order advection schemes [Selle et al. 2008] did not pay off. Our intuition here is that the synthetic data of the SDFs is noise-free and smooth. The alternative components mentioned previously typically introduced high-frequency details that in turn required more smoothing in later stages. As a result, we use simple first-order semi-Lagrangian advection, linear interpolation, and a regular Gaussian blur.

Table I. Parameters Used to Generate the Results of Section 7

Max. number of iterations	$l_{\max} = k_{\max} = 3$
Optical flow recursion threshold	$s_{\max} = 10$
Optical flow Laplacian weight	$\beta_S = 10^{-3}$
Optical flow Tikhonov weight	$\beta_T = 10^{-4}$
SDF distance range	$\gamma_{\max} = 40$
Scaling of SDF inputs	$\beta_{\text{image}} = -0.2/\gamma_{\max}$
Gaussian blur kernel	$\sigma_{\text{of}} = \sigma_{\text{proj}} = 4$
Projection narrow band	$\tau_{\text{proj}} = 4$
Conjugate gradient residual threshold	10^{-2}

The optical flow algorithm by default puts more emphasis on larger regions of the surface (the energy is minimized in a least-squares fashion, equally weighting every point in space). To make sure the first frame of the simulation is registered properly, we repeat it in time (five timesteps for our examples). Furthermore, we leave an empty region of 10% at all sides of the input, to give the optical flow the chance to freely push surfaces near the sides in- and outward.

We noticed that the extracted time slices for liquid SDFs can exhibit slight flickering. To alleviate this, we apply a temporal filter, for which we found the union of two adjacent extracted time slices to be the most efficient choice. Alternatively, the use of higher-order interpolations in time would be possible here.

7. RESULTS

In the following, we demonstrate our matching approach with several example inputs from smoke and liquid simulations. All timings were measured using an Intel Core i7 with 4GHz, and were generated with *mantaflow* [2016].

Smoke. As a first test case we consider two buoyant smoke clouds. The clouds have different initial positions, and only one of them directly interacts with a spherical obstacle in the scene. This leads to distinctly different shapes and motions of the two clouds. A successful 0.5 interpolation of these clouds with our method is shown in the middle of Figure 5. Generating the smoke volumes took only 0.56s per frame on average for this example. This extraction timing (as well as the following) includes all calculations necessary to extract the final volume to be displayed, excluding disk I/O.

We have used an isolevel of 10% of the maximal density to define the surface, and this value was used for all other smoke inputs in the following. The isosurfaces extracted in this way do not conserve volume, and can change significantly from one frame to the next (Figure 10, left and right). Despite these difficulties, our algorithm robustly recovers a match between the inputs, and the final deformation (Figure 10, middle) matches the target shape very well. The whole range of the complex interpolations calculated by our method is shown in Figure 12 for a static frame. Resolutions and timings for this and the following simulations can be found in Table II. Here, memory requirements are given in gigabytes, and timings are averaged values across a full simulation or FlOF solve.

Figure 11 shows a comparison of our method with other approaches using the same smoke inputs. On the left is a direct blend of two input volumes, which yields a clear duplication of the smoke

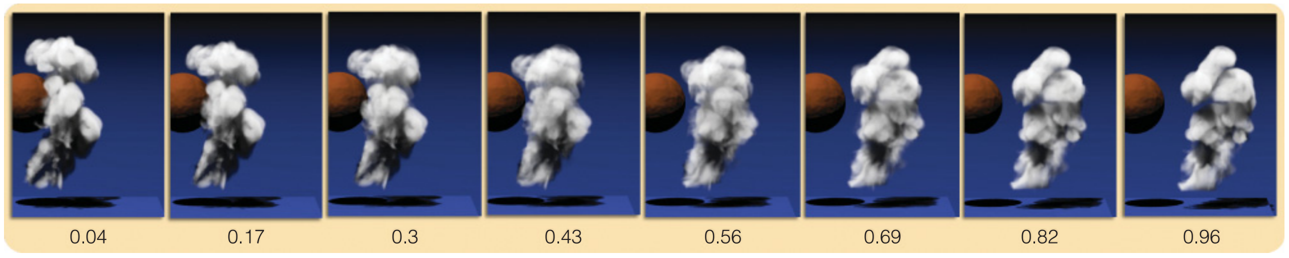

Fig. 12. The range of interpolations for the smoke setup of Figure 5 at a fixed point in time. The numbers below each image indicate the interpolation weight.

Table II. Simulation Resolutions and Timings

Fig.	Simulation		Deformation		
	Res.	Time	Res.	Mem.	Time
5	$200^2 \cdot 300 \cdot 150$	24m 2s	$50^2 \cdot 75^2$	4.96	6m 16s
11	”	”	$50^2 \cdot 75^2$	2.80	1m 27s$^{(+)}$
13	$200^3 \cdot 450$	30m 21s	$54^3 \cdot 81$	4.56	4m 24s
20	$240^3 \cdot 360$	25m 12s	$54^3 \cdot 81$	5.35	4m 53s
21	$300^3 \cdot 450$	34m 16s	$50^3 \cdot 75$	2.81	3m 07s

volumes. On the right, we applied a state-of-the-art hierarchical optical flow [Meinhardt-Llopis et al. 2013] to the two smoke volumes. The resulting deformation pushes the volumes in the right direction, but gives very undesirable stretching in space and time. For this optical flow solve, the CG residual threshold in Table II was lowered by 50% to ensure the match is as good as possible, and it uses identical input data as Figure 5. Our result, shown in the middle of Figure 11, successfully aligns the two clouds, giving a single rising column of smoke with an intermediate shape. Smaller wisps of smoke around the main cloud are moving along at no extra cost.

Liquid. Figure 13 demonstrates the FlOF algorithm and our interpolation scheme for a liquid example. The two inputs are simulations with the initial position of the drops interchanged. In the inputs the drops impact at different times, and the resulting splashes are asymmetric. Our deformation correctly retrieves a simultaneous impact for an interpolation with weight 0.5. A similar setup was simulated by Raveendran et al. [2014] using a significantly less detailed simulation that did not contain any drops or thin sheets. The accompanying video shows a direct comparison to illustrate the difference in surface complexity between this version and ours. In contrast to the explicit detection and removal of small features proposed for the ICP-based matching [Raveendran et al. 2014], our Eulerian SDFs contain averaged quantities in a broad band around the surface that we can match robustly. Additionally, our volumetric deformation fields can be applied to small-scale features near the surface without additional work. For a mesh-based approach, an additional extrapolation step would be required to extend the deformations into the volume. However, the ICP-based approach recovers the pure translational initial configuration of the two drops with higher accuracy.

For this example, the calculation of the 240^3 outputs with union blending (Section 5) took only $0.64s$ per frame on average. The largest deformations for this example have a magnitude of more than $150\Delta x$ in 4D space. This illustrates the large distances captured by our deformations.

For this liquid setup and the smoke setup of Figure 5, comparisons between our interpolated result and a new simulation at the intermediate position can be found in the supplemental video (several frames for the two-drop liquid test case are shown in Figure 22). As our method generates the output based on simulations with a different parameterization, it does not yield small-scale details that are identical to those of a full simulation, but it faithfully captures the behavior of the larger scales.

2D Parameters. To demonstrate interpolations within a 2D parameter space we have simulated the liquid setup shown in Figure 20. A liquid inflow on the left side is positioned at various depths and heights, some of which cause the liquid to hit the obstacle on the right wall, while other inflow positions partially hit or completely miss it. This leads to strongly varying splashes and waves in the inputs, the full sequences of which can be found in the supplemental materials. We use seven different input surfaces, yielding interpolations with six different triangular simplices covering a large space of fluid behavior. We calculated 12 deformations to cover this parameter space. As our algorithm works without requiring any user input, all deformations were generated automatically with the same set of parameters. Example outputs using a variety of deformation combinations can be found in Figures 1 and 20.

For this example, the obstacle on the right wall leads to an increased difficulty for our algorithm. We do not use this prior knowledge about the scene geometry for matching, and thus it is not guaranteed that the deformations do not push parts of the surface into the obstacle. Our results indicate what can be achieved without incorporating this information into the solve for six different simplices spanned by two deformations each. The large number of small-scale drops and splashes around the obstacle that are successfully matched and deformed highlight the complexity of the inputs our algorithm can deal with. However, in several instances flickering artifacts and surface break-up is noticeable. The following section will outline in more detail which parts of our algorithm are causing these.

The interpolations in a 2D parameter space led to surface extraction times of $0.87s$ per frame on average. As the operations involved purely consist of simple operations on regular grids, there is significant potential for optimization with parallel processing, for example, by using GPUs (which were not used in our implementation).

Smoke and Liquid. Finally, the example of Figure 21 matches an input of a falling drop of liquid with a blob of heavy smoke. The difference of the two phenomena demonstrates how our method can cope with challenging inputs: the drops have very different falling speeds, and while the liquid leads to splashes and waves, the smoke buoyancy gives rise to many complex swirling surfaces. These buoyant swirls translated into a complex isosurface, that was successfully matched to the liquid one by our algorithm. Once the match is calculated (treating both inputs as density data), we can easily interpolate any in-between behavior. This controllable transition of behavior is clearly beyond the scope of a regular fluids

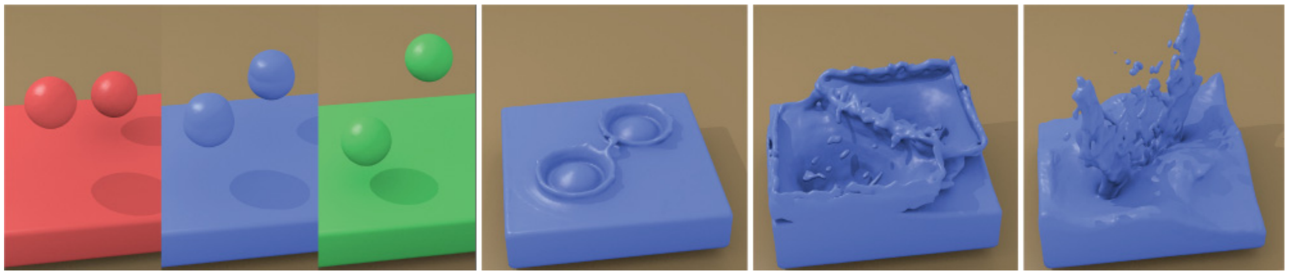

Fig. 13. An interpolation of two liquid simulations. The inputs (in red and green on the left), have different initial drop positions. Our 0.5 interpolation (in blue) aligns them in space and time so that the impact of the drops coincides. The three images on the right show later frames of our interpolated version.

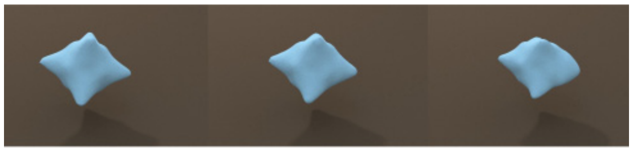

Fig. 14. Matching a single star-shaped surface with increasing distances. From left to right: a distance of 10, 20, and 30 cells, respectively (with an interpolation weight of 0.4). For the largest distance the tips of the star are not fully recovered anymore.

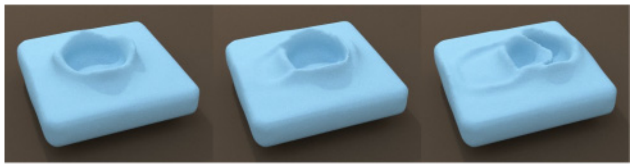

Fig. 15. The resulting surfaces for deformations computed with two falling drops with increasing distances (20, 30, and 40 cells from left to right). For a distance of 40 cells (right), the source splash is still visible after applying the deformation.

solver. Our interpolation also conveniently blends between the differing viscosities of the FLIP and Eulerian-only simulations, and illustrates that our method is agnostic to the type of input: arbitrary parameters could be varied, from scene geometry to physical parameters.

8. APPLICABILITY AND LIMITATIONS

Our approach clearly does not work for arbitrary pairs of inputs: if the two surfaces share no similarities, we can compute a deformation, but the resulting interpolations will yield unexpected or undesirable results. As it is difficult to explicitly specify a region of applicability, we present a series of tests in the following to illustrate in which cases our method yields the expected results, and where it is likely to fail. For all of the following failure cases, a straightforward fix is to insert additional data points, but we will discuss alternative directions for future extensions where appropriate. We distinguish limitations of computing deformations with our FlOF algorithm, and limitations arising from the subsequent interpolation. Full animations of inputs and outputs for all cases in the following can be found in the supplemental video. Unless specified otherwise, the following tests use datasets with a resolution of 64^4 and deformations of 50^4.

Deformation. Next, we will show the results of applying a single deformation to sequences that are as simple as possible, such that the quality of the resulting deformation can be evaluated visually.

The matching with our algorithm is inherently based on closest distances in four-dimensional Euclidean space. This can lead to unexpected correspondences in certain situations. For example, for two spherical shapes that are more than their radius apart, a closest distance will create a correspondence between two opposing sides of the spheres. For such a case, we have to rely on the smoothness constraint across multiple scales in the optical flow solve to match the correct sides of each sphere. As a consequence, our method does not always recover an ideal rigid translation for large distances of the inputs. This effect is noticeable for the spherical drop of Figure 13, and the cylindrical stream of Figure 20. Additionally, if the inputs share little or no similarities, the deformations will try to match pieces in proximity, but not necessarily the whole target shape. Note that the timestep of the input data also influences the relative scaling of spatial and temporal distances.

We illustrate the behavior of our algorithm in Figure 14, where we match two identically moving star shapes with increasing spatial distances. As the distance grows to more than ca. 25% of the domain size, the deformation has trouble resolving the full shape of the target. As can be seen in the supplemental video, the final shape is matched quite well, but the noneven distribution of surface points (i.e., a nonrigid deformation of the input) leads to a loss of features for interpolation weights less than 1.

Moving to slightly more complicated inputs, the surfaces shown in Figure 15 exhibit topology changes when a drop hits a basin of liquid. With growing distances between the inputs, the deformation starts to have difficulties resolving the ambiguous surfaces around the time of impact. For both of the preceding cases it could help to automatically detect and match feature points (such as a topology change). These matches could then be enforced as constraints in the FlOF solve to better match important features of the inputs.

Overall, our method is quite robust with respect to matching small components, and fast moving objects, as long as they are represented at the resolution of the FlOF solve. This is illustrated with the setup of Figure 16, where the size of a falling drop is continually decreased while the timestep of the simulations increases. For each different set of parameters we match two 4D surfaces where the drops have a fixed spatial distance of 20 cells. The translational component is recovered even for small and quickly moving drops, the most difficult part being the topology change. We have used sizes of 8, 5, and 3 cells, and timesteps of 2.0, 2.75, and 3.25, respectively (for reference, the drop example of Figure 15 uses a size of 10 and a timestep of 0.85). The version with a timestep of 2.75 is shown in Figure 16.

A more complex example of a failure case for our matching algorithm is shown in Figure 17. This setup is based on the liquid

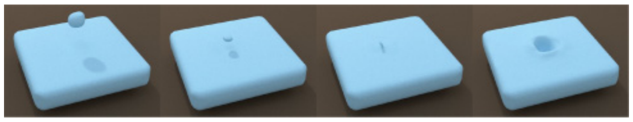

Fig. 16. A deformation of a small, fast moving drop over time. While the main motion is recovered, the surface can start to break up during the impact for the fast moving datasets.

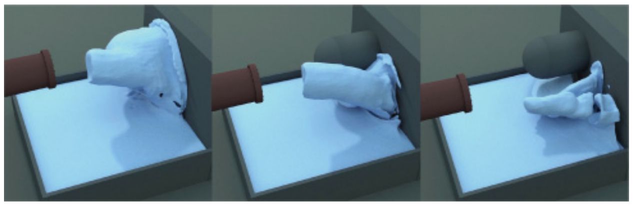

Fig. 17. A more complex failure case: we match a simulation with an inflow in the back with one that is moved further and further to the front. The images above show stills at the same time for an interpolation weight of 1. The target simulations had inflow shifts of 18%, 32%, and 46% of the domain size (from left to right).

stream from Figure 20 (parameters can be found in Table II). A large distance between source and target shape results in a deformation that does not fully retrieve the target shape when it is applied. This is particularly visible for the cylindrical stream of the inflow, in the right image of Figure 17: the cylinder is only partially present in the deformed output.

As a rule of thumb, distances larger than ca. 25% of the domain size can lead to suboptimal deformations with our algorithm. This estimate is partially a result of our implementation, which coarsens the optical flow solve until a minimum resolution is reached ($s_{\max}$ in Algorithm 1). With a different coarsening strategy, for example, enlarging the domain along with the coarsening, potentially larger distances could be successfully matched. A second aspect to take into account are relative distances in the datasets. As optical flow matches surface in a nearest-neighbor fashion, a match across a large distance is only made if no suitable surface in closer proximity is found. For the datasets shown, we found that the 25% rule prevented most ambiguities for the large-scale features of the surface.

Another difficulty in Figure 17 is the presence of the obstacle on the right, which leads to substantially different liquid motions and surface shapes for the splash against the wall. For the datasets with large distances this leads to flickering and dissolving surfaces. While using higher resolutions for the output reduces the chance of surface flickering, a better way to prevent these artifacts would be to introduce a third data point in between the two existing ones, reducing the relative distances of the inputs. This would help to establish the desired correspondences based on closest distances, and could potentially be done automatically based on the final error measurement after the FlOF solve.

Interpolation. Our interpolation step combines two or more deformations to produce an output. While the previous paragraph illustrated the behavior of a single deformation, we now show interpolations using two deformations and two input sequences (with per frame resolutions of 192^3).

Figure 18 illustrates the advantages and disadvantages of the union blending from Section 5. On the left a simple union of the undeformed inputs can be seen for reference. In the middle, a deformation is applied with strength 0.5 to both inputs, and the deformed

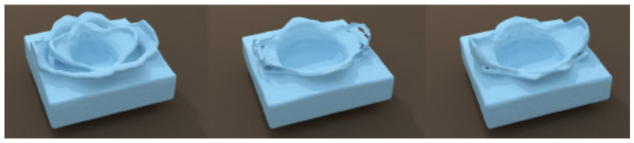

Fig. 18. Three different versions of an interpolation of two falling drops for weight 0.5. From left to right: a union of both inputs without deformation, a linear interpolation with deformation, and ours.

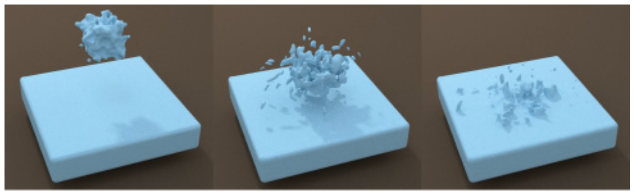

Fig. 19. Several frames over time of an especially difficult matching problem: a collection of randomly moving small droplets.

surfaces are averaged to produce an interpolation at the midpoint. While the splashes align, the thin sheets break up, leading to flickering effects during animations. The version on the right uses the same deformation, but interpolates both surfaces with the union-blending approach. This significantly reduces the likelihood of thin structures breaking up. However, it can lead to duplicate surfaces, as can be seen, for example, on lower left side of the thin sheet of Figure 18 (right). For smoke simulations, such misalignments can lead to ghosting artifacts at the sides of clouds.

A particularly tough case for matching and interpolation is shown in Figure 19: here two sets of randomly moving drops are registered with each other (with a spatial distance of 20 cells). In contrast to Figure 16 the individual drops are not well represented anymore for the optical flow solve. The accompanying animation shows how an increasingly strong motion leads to difficulties resolving the different features of the two inputs, and results in flickering motions of the droplets. While our algorithm still recovers the overall translation, the independently moving drops cannot be resolved. A similar effect is noticeable in some of the interpolations of Figure 20. These artifacts could be reduced by increasing the resolution of the FlOF solve, or by switching to a particle representation for the drops. Additionally, our algorithm could be used in conjunction with one or more postprocessing steps, for example, to generate spray particles and other secondary effects based on the interpolated surfaces.

Note that combining deformations is easier in a Lagrangian setting, for example, for meshes. However, we believe that the gains in robustness and surface quality outweigh the additional computational cost for aligning Eulerian deformations.

9. DISCUSSION

Our approach relies on a reasonable isosurface thresholding when working with a smoke simulation. However, we had no problem selecting a suitable threshold for our tests. For scenarios with lots of uniform densities, it would also be possible to invest more computational work to initialize and track separate implicit surfaces that could then be matched by our algorithm. Additionally, an interesting venue for future work would be the inclusion of Eulerian advection schemes that propagate information forward (instead of backward) [Lentine et al. 2011]. These algorithms are typically more complicated, but could circumvent some of the alignment problems discussed in Section 4.

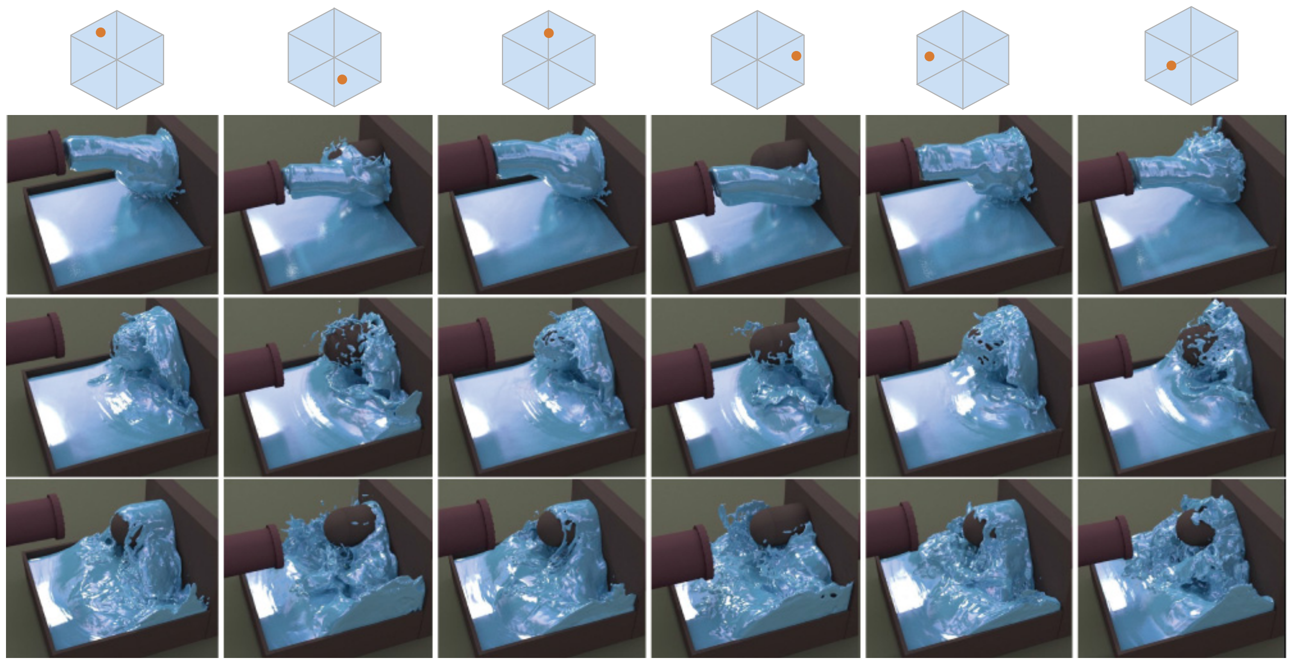

Fig. 20. Six examples of interpolations at different points of a 2D parameter space (shown at the top) for a liquid setup. The outputs exhibit strongly differing behavior, with distinct splashes and wave motions depending on where in the scene the inflow stream hits the obstacle and pool.

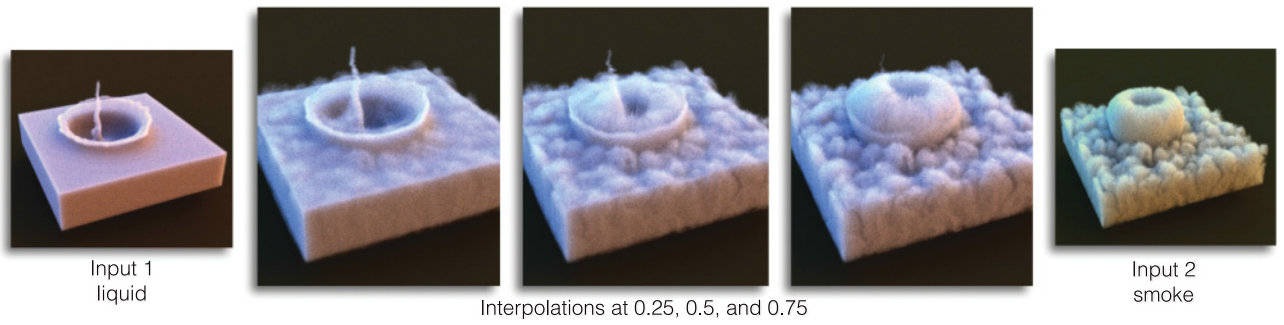

Fig. 21. On the far left and right the inputs are shown: a drop of liquid, and a (cold) blob of smoke, each impacting a basin of liquid and smoke, respectively. The different interpolations in the center (with a resolution of 300^3) illustrate our matching of the two different phenomena.

For larger datasets, hard-disk access can become a noticeable component of the runtimes. If this becomes a bottleneck, compression schemes to reduce the size of the stored volumes could be introduced. Also, our interpolation would need to be changed to guarantee smooth transitions across simplex boundaries, and we defer an extension of our interpolation scheme with bidirectional deformations along simplex edges to future work. For interpolations with more than three dimensions, it is also nontrivial to generate simplicial tessellations. While regular data points could be manually tessellated for higher dimensions, arbitrary sample locations would be tricky to deal with.

Lastly, we have ignored the underlying physics during the optimization procedure. Our intention was to explore how far a purely data-driven optical flow solve could be taken in this setting. However, the algorithm could be readily combined with other optimization schemes [Gregson et al. 2014] to incorporate additional physical constraints. For example, it will be highly interesting to include constraints for mass conservation of the deformed motion from one frame to the next.

10. CONCLUSIONS

We have presented an unconventional way to employ optical flow for interpolating space-time data of fluid simulations. The resulting algorithm can find complex 4D registrations without user input, and is able to match phenomena as different as the smoke plume and liquid drop in Figure 21.

This could be highly interesting not only for fluid animations, but also for other types of data, such as character animations without temporally coherent meshes (connectivity information quickly gets lost in the different stages of a production pipeline). Here, additional constraints such as piecewise rigidity will be interesting avenues for future work. We also envision our work to be very useful in the context of precomputing complex interactive effects [Stanton et al.

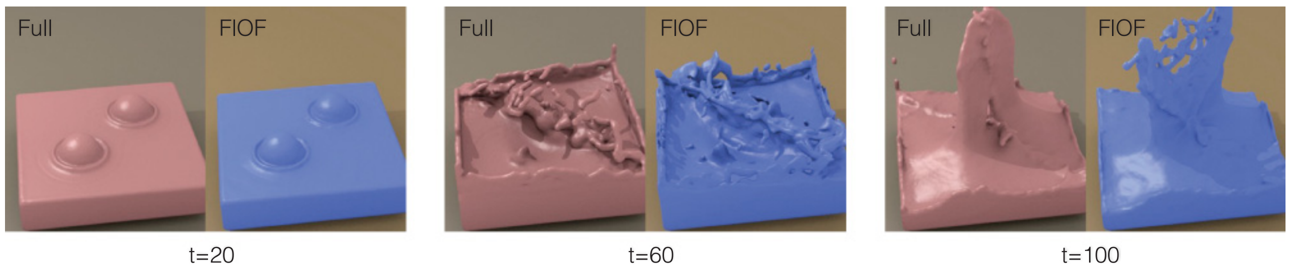

Fig. 22. Several frames comparing our interpolated result at 0.5 with a full simulation with a corresponding initial setup (i.e., drops at same height). Each image shows the full simulation on the left, ours on the right. While small-scale structures deviate, our interpolation recovers the large-scale structures.

2014], and we are currently exploring its application in real-time settings. The simplicity of the calculations for surface extraction should make interactive frame rates on mobile devices possible.

Finally, it is an exciting outlook to have tools that automatically analyze and register large collections of space-time datasets. This would open up numerous interesting applications for offline as well as real-time effects beyond data-driven fluid simulations.

APPENDIX

Weights for 2D Interpolation

The following equations outline the weight calculations for the interpolation of Figure 20, with the notation of Figure 9. We assume $\mathbf{x} = (x_1, x_2)^T$ in barycentric coordinates; the two weights are given relative to $\mathbf{\Psi}_1$ in Figure 9:

$$
\begin{aligned}
(\omega_1, \omega_{1\cup 2}, \omega_{1\cup 3})^T &= H\Big[\mathbf{x}, (1,0)^T, (\tfrac{1}{2}, \tfrac{1}{2})^T, (\tfrac{1}{2}, 0)^T\Big],\\
(\omega_2, \omega_{1\cup 2}, \omega_{2\cup 3})^T &= H\Big[\mathbf{x}, (0,1)^T, (\tfrac{1}{2}, \tfrac{1}{2})^T, (0, \tfrac{1}{2})^T\Big],\\
(\omega_3, \omega_{1\cup 3}, \omega_{2\cup 3})^T &= H\Big[\mathbf{x}, (0,0)^T, (\tfrac{1}{2}, 0)^T, (0, \tfrac{1}{2})^T\Big],\\
(\omega_{1\cup 2}, \omega_{1\cup 3}, \omega_{1\cup 2\cup 3})^T &= H\Big[\mathbf{x}, (\tfrac{1}{2}, \tfrac{1}{2})^T, (\tfrac{1}{2}, 0)^T, (\tfrac{1}{3}, \tfrac{1}{3})^T\Big],\\
(\omega_{1\cup 2}, \omega_{2\cup 3}, \omega_{1\cup 2\cup 3})^T &= H\Big[\mathbf{x}, (\tfrac{1}{2}, \tfrac{1}{2})^T, (0, \tfrac{1}{2})^T, (\tfrac{1}{3}, \tfrac{1}{3})^T\Big],\\
(\omega_{1\cup 3}, \omega_{2\cup 3}, \omega_{1\cup 2\cup 3})^T &= H\Big[\mathbf{x}, (\tfrac{1}{2}, 0)^T, (0, \tfrac{1}{2})^T, (\tfrac{1}{3}, \tfrac{1}{3})^T\Big].
\end{aligned}
$$

ACKNOWLEDGMENTS

We would like to thank Karthik Raveendran, Chris Wojtan, and Greg Turk for the numerous discussions about blending and interpolating liquids. Additionally, thanks to James Gregson, Ivo Ihrke, Wolfgang Heidrich, and Daniel Cremers for insights and discussions about optical flow. Thanks also to Ted Kim and Tiffany Inglis for proofreading early versions, and to Rachel Chu for help with the video.

Finally, thanks to the anonymous reviewers for the many helpful suggestions improving the article.

REFERENCES

Morten Bojsen-Hansen, Hao Li, and Chris Wojtan. 2012. Tracking surfaces with evolving topology. *ACM Transactions on Graphics* 31, 4, Article 53 (July 2012), 10 pages.

Robert Bridson. 2016. *Fluid Simulation for Computer Graphics*. AK Peters/ CRC Press.

Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. 1996. *Metro: Measuring Error on Simplified Surfaces*. Technical Report. Paris, France.

Daniel Cremers and Stefano Soatto. 2003. A pseudo-distance for shape priors in level set segmentation. In *Proceedings of the 2nd IEEE Workshop on Variational, Geometric and Level Set Methods in Computer Vision*. 169–176.

Jan Ehrhardt, René Werner, Dennis Säring, Thorsten Frenzel, Wei Lu, Daniel Low, and Heinz Handels. 2007. An optical flow based method for improved reconstruction of 4D CT data sets acquired during free breathing. *Medical Physics* 34, 2 (2007), 711–721.

Nick Foster and Dimitri Metaxas. 1996. Realistic animation of liquids. *Graphical Models and Image Processing* 58, 5 (Sept. 1996), 471–483. DOI:http://dx.doi.org/10.1006/gmip.1996.0039

James Gregson, Nils Thuerey, Ivo Ihrke, and Wolfgang Heidrich. 2014. From capture to simulation—Connecting forward and inverse problems in fluids. *ACM Transactions on Graphics* 33 (4) (August 2014), 10.

Berthold Horn and Brian Schunck. 1981. Determining optical flow. *Artificial Intelligence* 17, 1 (1981), 185–203.

Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. 2014. SPH fluids in computer graphics. In *Eurographics—State of the Art Reports*. 21–42.

Won-Ki Jeong and Ross T. Whitaker. 2008. A fast iterative method for eikonal equations. *SIAM Journal of Scientific Computing* 30, 5 (2008), 2512–2534.

Theodore Kim and John Delaney. 2013. Subspace fluid re-simulation. *ACM Transactions on Graphics* 32, 4, Article 62 (July 2013), 9 pages.

Theodore Kim, Nils Thürey, Doug James, and Markus Gross. 2008. Wavelet turbulence for fluid simulation. *ACM Transactions on Graphics* 27, 3, Article 50 (August 2008), 1–6.

Lubor Ladicky, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, Markus Gross, and others. 2015. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 199.

Michael Lentine, Mridul Aanjaneya, and Ronald Fedkiw. 2011. Mass and momentum conservation for fluid simulation. In *Proceedings of Symposium on Computer Animation*. ACM, 91–100.

Yangyan Li, Xiaochen Fan, Niloy J. Mitra, Daniel Chamovitz, Daniel Cohen-Or, and Baoquan Chen. 2013. Analyzing growing plants from 4D point cloud data. *ACM Transactions on Graphics* 32, 6, Article 157 (Nov. 2013), 10 pages.

Enric Meinhardt-Llopis, Javier Sánchez Pérez, and Daniel Kondermann. 2013. Horn-Schunck optical flow with a multi-scale strategy. *Image Processing On-Line* 2013 (2013), 151–172.

M. Müller, D. Charypar, and M. Gross. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of Symposium on Computer Animation*. 154–159.

Michael B. Nielsen, Andreas Söderström, and Robert Bridson. 2013. Synthesizing waves from animated height fields. *ACM Transactions on Graphics* 32, 1, Article 2 (Feb. 2013), 9 pages.

Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. 2013. Interactive localized liquid motion editing. *ACM Transactions on Graphics (SIGGRAPH Asia 2013)* 32, 6 (Nov. 2013).

Nikos Paragios, Mikael Rousson, and Visvanathan Ramesh. 2003. Non-rigid registration using distance functions. *Computer Vision and Image Understanding* 89, 2 (2003), 142–165.

Tiberiu Popa, Ian South-Dickinson, Derek Bradley, Alla Sheffer, and Wolfgang Heidrich. 2010. Globally consistent space-time reconstruction. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1633–1642.

Karthik Raveendran, Nils Thuerey, Chris Wojtan, and Greg Turk. 2014. Blending liquids. *ACM Transactions on Graphics* 33, 4 (August 2014), 10.

Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. 2008. An unconditionally stable MacCormack method. *Journal of Scientific Computing* 35, 2–3 (June 2008), 350–371.

Andrei Sharf, Dan A. Alcantara, Thomas Lewiner, Chen Greif, Alla Sheffer, Nina Amenta, and Daniel Cohen-Or. 2008. Space-time surface reconstruction using incompressible flow. *ACM Transactions on Graphics* 27, 5, Article 110 (Dec. 2008), 10 pages.

Lin Shi and Yizhou Yu. 2005. Taming liquids for rapidly changing targets. In *Proceedings of Symposium on Computer Animation*. 229–236.

Jos Stam. 1999. Stable fluids. In *Proceedings of ACM SIGGRAPH*. 121–128.

Jos Stam and Ryan Schmidt. 2011. On the velocity of an implicit surface. *ACM Transactions on Graphics* 30, 3 (2011), 7.

Matt Stanton, Ben Humberston, Brandon Kase, James O'Brien, Kayvon Fatahalian, and Adrien Treuille. 2014. Self-refining games using player analytics. *ACM Transactions on Graphics* 33, 4 (2014), 9.

Deqing Sun, Stefan Roth, and Michael J. Black. 2014. A quantitative analysis of current practices in optical flow estimation and the principles behind them. In *International J ournal of Computer Vision* 106, 2 (Jan. 2014), 115–137.

Nils Thuerey, Richard Keiser, Ulrich Ruede, and Mark Pauly. 2006. Detail-preserving fluid control. In *Proceedings of Symposium on Computer Animation*. 7–12.

Nils Thuerey and Tobias Pfaff. 2016. MantaFlow. (2016). http://mantaflow.com.

Adrien Treuille, Andrew Lewis, and Zoran Popović. 2006. Model reduction for real-time fluids. *ACM Transactions on Graphics* 25, 3 (July 2006), 826–834.

Sundar Vedula, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade. 1999. Three-dimensional scene flow. In *Proceedings of Computer Vision*, Vol. 2. IEEE, 722–729.

Andreas Wedel and Daniel Cremers. 2011. *Stereoscopic Scene Flow for 3D Motion Analysis*. Springer.

Zexiang Xu, Hsiang-Tao Wu, Lvdi Wang, Changxi Zheng, Xin Tong, and Yue Qi. 2014. Dynamic hair capture using spacetime optimization. *ACM Transactions on Graphics* 33, 6, Article 224 (Nov. 2014), 11 pages.

C. Zach, T. Pock, and H. Bischof. 2007. A duality based approach for realtime TV-L1 optical flow. In *Proceedings of the 29th DAGM Conference on Pattern Recognition*. 214–223.

Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Transactions on Graphics* 24, 3 (2005), 965–972.

Darko Zikic, Ali Kamen, and Nassir Navab. 2010. Revisiting Horn and Schunck: Interpretation as Gauss-Newton optimisation. In *Proceedings of the British Machine Vision Conference*. 1–12.

Received July 2015; revised June 2016; accepted June 2016