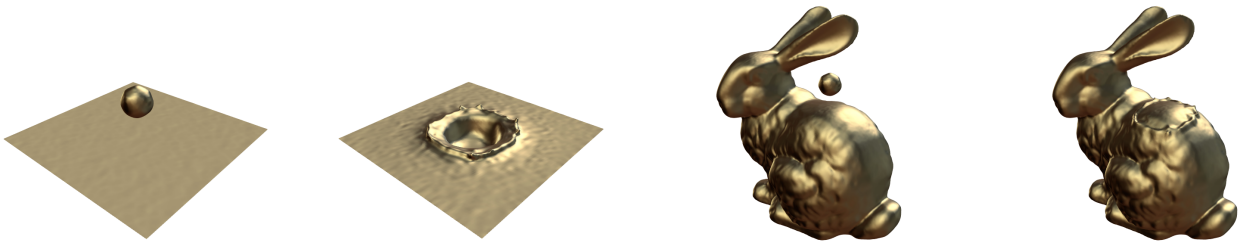
Space-time sculpting of liquid animation

Pierre-Luc Manteaux[1†] Ulysse Vimont[1†] Chris Wojtan[2] Damien Rohmer[3] Marie- Paule Cani[1]
University Grenoble-Alpes, CNRS (LJK) and Inria[1] IST Austria[2] CPE Lyon, University Lyon[3] *

† Pierre-Luc Manteaux and Ulysse Vimont are joint first authors



(a) Initial simulation at time t_0 (b) Initial simulation at time t_1 (c) Final animation at time t_0 (d) Final animation at time t_1

Figure 1: *From an initial simulation of a falling drop (a, b) space-time features can be extracted and pasted back into an other mesh, generating a new animation (c, d).*

Abstract

We propose an interactive sculpting system for seamlessly editing pre-computed animations of liquid, without the need for any re-simulation. The input is a sequence of meshes without correspondences representing the liquid surface over time. Our method enables the efficient selection of consistent space-time parts of this animation, such as moving waves or droplets, which we call *space-time features*. Once selected, a feature can be copied, edited, or duplicated and then pasted back anywhere in space and time in the same or in another liquid animation sequence. Our method circumvents tedious user interactions by automatically computing the spatial and temporal ranges of the selected feature. We also provide space-time shape editing tools for non-uniform scaling, rotation, trajectory changes, and temporal editing to locally speed up or slow down motion. Using our tools, the user can edit and progressively refine any input simulation result, possibly using a library of pre-computed space-time features extracted from other animations. In contrast to the trial-and-error loop usually required to edit animation results through the tuning of indirect simulation parameters, our method gives the user full control over the edited space-time behaviors.

Keywords: fluid animation, sculpture, space-time editing

Concepts: •**Computing methodologies** → **Animation; Shape analysis;**

1 Introduction

Due to advances in fluid simulation methods over the last two decades, liquid animation has become commonplace in 3D animation productions [Van Opstal et al. 2014; Reisch et al. 2016]. The animations can be either highly realistic — for example showing plausible fluid dynamics and interactions with obstacles — or they can exhibit a more expressive behavior to convey specific artistic intentions. In both cases, it is essential for the artist to be able to control the simulation in order to achieve their goals.

*{ pierre-luc.manteaux | ulysse.vimont | marie-paule.cani }@inria.fr
wojtan@ist.ac.at
damien.rohmer@cpe.fr

Generally, the simulation control is achieved through the careful setting of a large number of parameters such as initial conditions, boundary conditions, viscosity, or external forces. Several reasons make the tuning of these parameters especially difficult. First, they only offer indirect control over the animation, which makes them quite non-intuitive. Second, it is usually not possible to have interactive visual feedback when modifying the parameters, due to the high computational cost of liquid simulation. Third, the inherently non-linear nature of fluid behavior makes it difficult to transfer parameter values from a low to a high resolution simulation. In consequence, achieving a desired effect requires a tedious trial-and-error loop, where computation is restarted multiple times from scratch with different parameters. In many cases, this process does not allow tight control over a sequence of waves and splashes with specific magnitudes or shapes and occurring in a specific order.

In this work, we propose a significantly different approach. Instead of controlling a simulation, we propose an interactive sculpting system enabling to edit pre-computed liquid animations. Our system is based on a copy/edit/paste approach: The user can select coherent and visually important space-time parts of a liquid animation, such as waves or droplets, that we call *space-time features*; These space-time features can then be edited in both space and time in order to change their size, orientation, trajectory or speed. Finally, the edited space-time feature can be inserted into any destination animation at a specific position and time set by the user.

To enable the use of arbitrary liquid animations computed using varying simulation techniques, we based our editing framework on generic inputs; our method allows input mesh sequences without point-wise correspondences between frames, and with arbitrary changes of topological genus between two consecutive time steps. Also, we focused on three requirements to make our method useful in realistic cases. First, the selection of the effect in the original simulation must be as simple and straightforward for the user as possible. Therefore, once space-time features have been computed, the user can select them using a simple click on the surface. Secondly, pasting the selected effect onto the final animation should be handled automatically, with seamless adaptation of the pasted fluid effect to the destination surface. Finally, the pipeline of selection, copy, edit, and paste steps should be computed efficiently in order to enable interactive user feedback.

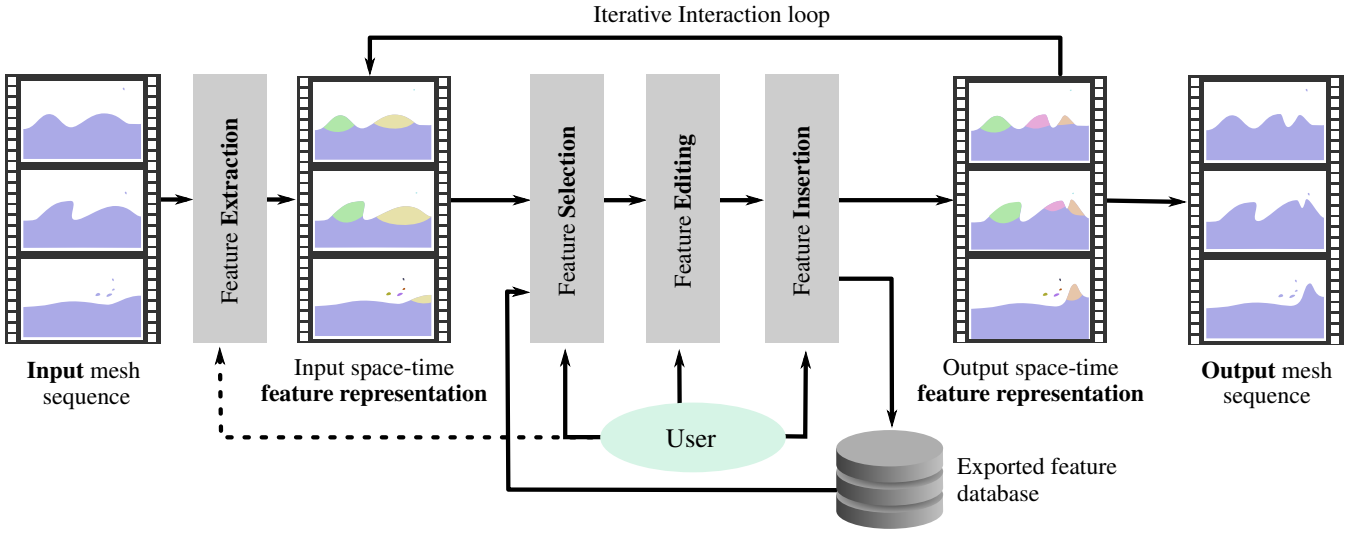


Figure 2: *Pipeline of our method: An input fluid animation is given as a mesh sequence. It is pre-processed into a higher-level space-time feature representation. This representation allows the user to iteratively select features from the animation and edit them before inserting them back to the animation. Alternatively, features can be saved and re-imported in this animation or a different one.*

The key contributions of our work are as follows:

- A semi-automatic method to tag salient regions in a liquid animation.
- An algorithm that extracts coherent *space-time features* from a mesh sequence with tagged vertices.
- A *space-time feature* representation independent from the original animation.
- A set of editing operations that allow the extraction, manipulation, and insertion of *space-time features* into an animation.

The remainder of this paper is organized as follows: Section 2 discusses previous works aimed at manipulating liquid animations; Section 3 presents our solution to this problem; Section 4 explains how *space-time features* are computed; Section 5 deals with the *space-time features* representation; Section 6 details the tools we offer for manipulating *space-time features*; Section 7 shows results obtained with our method; Section 8 draws the limits of our approach; and Section 9 concludes and gives some perspectives on future work.

2 Related work

Since the introduction of *space-time constraints* [Witkin and Kass 1988], direct editing of simulation has been addressed in the contexts of rigid bodies [Popović et al. 2000; Chenney and Forsyth 2000; Twigg and James 2007] and of deformable objects [Wojtan et al. 2006; Barbič et al. 2009; Barbič et al. 2012; Schulz et al. 2014; Li et al. 2014]. However, few works address fluid surfaces, due to their constantly changing shape and topology that makes the output geometry inaccessible to standard deformation tools. This section focuses on two main methods for designing fluid animations: by *controlling the simulation*, and by *editing the animation*.

As our method only takes as input a sequence of meshes, it is completely independent from the fluid simulator. A detailed survey of fluid simulation techniques is therefore out of the scope of the paper.

2.1 Fluid simulation control

The general approach for controlling a simulation is based on trial and error. As mentioned in the introduction, this process has severe drawbacks which make the design of fluid animation especially tedious. To overcome these limitations, several methods propose to guide the fluid behavior by using geometric proxies which are easier to control than the high resolution fluid simulation data itself. For example, artists can use a triangle mesh to specify a target shape for the fluid by adding artificial attraction forces based on the distance to the mesh surface. Such approaches have been successfully developed to drive smoke [Fattal and Lischinski 2004; Hong and Kim 2004; Shi and Yu 2005b] and liquid simulations [Shi and Yu 2005a; Raveendran et al. 2012]. These meshes can also define specific key-frames to control an animation [Treuille et al. 2003; McNamara et al. 2004]. Fluid trajectories, on their sides, may be controlled with user-defined velocity fields [Kim et al. 2006], distance fields [Yang et al. 2013], or specific control particles [Thürey et al. 2006; Madill and Mould 2013].

Taking this strategy further, the attracting surface itself can defined a low-resolution fluid simulation. To achieve this, the artist quickly sets up a coarse simulation and uses the output geometry to guide the main features of a full resolution simulation. Several approaches modify a high-resolution smoke simulation using optimization [Nielsen et al. 2009; Nielsen and Christensen 2010], patterns extracted as skeleton [Yuan et al. 2011], or sparse sampling [Huang and Keyser 2013]. For liquid simulations, Nielsen and Bridson [2011] propose to restrict the high resolution simulation to a thin layer around a guiding coarse animation.

Although each of these approaches are able to successfully guide a fluid simulation, they do not enable direct control of the resulting fluid. Designing precise timing or feature scaling would therefore still require iterative trial-and-error steps to converge toward a desired animation. Few attempts have been made to enable direct control on the simulation. Schpok et al. [2005] propose to extract and parametrize features such as vortices, uniform advection, sinks, and sources to allow the user to modify the parameters in a smoke simulation. In the context of liquid simulation, Pan et al. [2013] propose a method to deform wave shapes by sketching their profiles. This approach enables direct spatial deformation but does not

allow temporal editing, and the simulation needs to be re-computed from the modified frame onwards.

Alternatively, procedural tools enable faster editing loops. Unfortunately, they are also based on indirect parameters setting and are often limited to overly restrictive fluid models such as large open ocean surfaces [Hinsinger et al. 2002; Tessendorf 2004; Jeschke and Wojtan 2015; Horvath 2015].

2.2 Fluid animation editing

Animation editing methods aim to directly modify the result of a simulation without the need to re-simulate. In contrast with simulation control, animation editing is often a faster approach to slightly modify an existing animation. Very few works have been proposed for the edit of fluid animations. For smoke animations, Pighin et al. [2004] propose to parametrize density and temperature fields from the simulation using advected radial basis function. The parametrized data are then deformed using a trajectory-based editing tool. For liquid animations, Raveendran et al. [2014] propose a semi-automatic method to match two animations and smoothly blend between them. Their method can quickly produce in-between frames and explore slight modifications of an animation. To our knowledge, there are no tools allowing a user to select and edit space-time regions of a fluid animation interactively.

3 Overview

In this work, we focus on editing liquid animations. To be independent from the simulation method, we take as input a sequence of meshes without any correspondences between the mesh vertices from one frame to another. Due to the arbitrary topology of the meshes and to the temporal coherence to be maintained for numerous geometric details, editing each frame with a shape modeling tool would represent a tremendous amount of work. Instead, we propose to manipulate a higher level representation of the liquid animation that we call *space-time features*. A space time feature is a sub-part of the animation, i.e. a sequence of sub-parts of the liquid surface.

Our editing pipeline generalizes standard sculpting tools [Ferley et al. 2000]: cut/copy/edit/paste. It is made of three steps which are illustrated in Figure 2. The first step extracts space-time features from the animation. As these features represent regions that deform over time, it would be too tedious for a user to define them by hand. We propose a semi-automatic method to detect salient regions in a liquid animation from which space-time features will be automatically computed. The user can then easily select them using picking: a click at a specific location at a given frame in time results in the automatic selection of the associated feature with its full range in space and time. The second step computes representations of the selected space-time features that are independent from the input animation. They enable space-time features to be transferred from one animation to another. Finally, the last step consists of editing the space-time features and pasting them back into an animation.

4 Feature extraction

In the feature extraction step, our method defines the space-time features that the user would like to manipulate. This process is divided into three steps, as described in Figure 3: detection, segmentation, and aggregation. While detection is semi-automatic (it is interleaved with user interaction to define customized regions of interest throughout the animation), segmentation and aggregation are fully automatic.

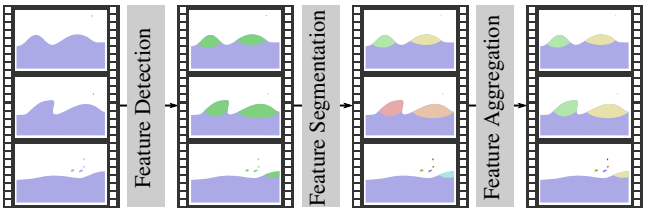


Figure 3: *Feature extraction process, from left to right: an initial mesh sequence representing a fluid animation is subjected to a feature detection process, followed by a segmentation step, which results in a frame feature representation. A final aggregation step allows to build a temporally coherent feature structure.*

Notation The input of our method is a mesh sequence over the time steps t that we note $M = (M^t)$, where M^t is a manifold triangular mesh. We note $T(.)$ the temporal length (i.e. the number of frames) and $L(.)$ the characteristic spatial length (i.e. the length of the diagonal of the spatial bounding box) of any space-time sequence (mesh sequence or feature). Given a triangular mesh X, we call N_X the set of its vertices and P_X the set of its faces. A vertex can carry attributes. We note $A(n, X)$ the value of the attribute A at the vertex n of the mesh X. In the following, we will note $pos(n, X)$, $norm(n, X)$, and $curv(n, X)$ for positions, normal, and curvature respectively. $\Delta_A(n, X)$ designates the Laplace-Beltrami operator [Botsch et al. 2007] applied to the attribute A at vertex n of the mesh X. A comprehensive list of notations can be found in Appendix A.

4.1 Detection

The detection phase aims at defining a sequence of regions of interest $R = (R^t)$ on M. A region R^t is represented as a set of vertices of M^t; we call this structure a *mesh part*.

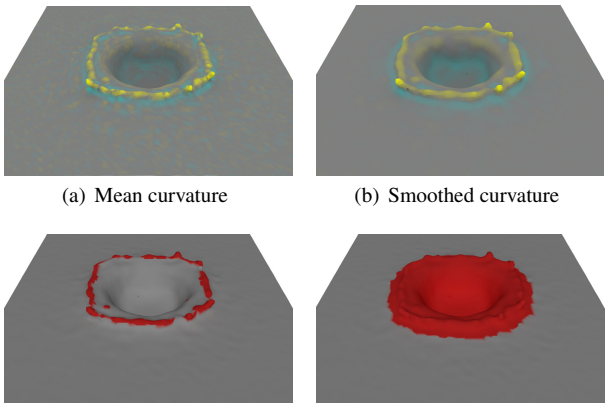
To let the user easily and intuitively define R, we propose a semi-automatic tool. This tool is based on two key components that we describe in detail below: curvature analysis and topological filtering. Combined together they let the user define R in a coarse-to-fine manner: First, curvature analysis is used to automatically detect salient features at each frame and initialize R. Then, topological filtering allows to interactively adjust R. We also added a painting tool that allows the user to fine-tune each R^t if needed by locally removing or adding vertices from R by clicking.

Multi-resolution curvature analysis. We chose a curvature criteria to extract features as it is a natural asset for detecting waves and ripples in liquid animations. Moreover, the intimate relationship between surface curvature and liquid surface dynamics had already led previous work to use curvature as a tool to enrich liquid simulations, for example with splashes [Takahashi et al. 2003], foam [Ihmsen et al. 2012] and textures [Narain et al. 2007].

Curvature is computed at each vertex n of the animation meshes M^t using the following formula:

$$curv(n, M^t) = norm(n, M^t) \cdot \Delta_{pos}(n, M^t) \qquad (1)$$

Vertices are colored with respect to their curvature magnitude, enabling the user to interactively observe the curved regions and their deformations on the fluid surface while playing the animation (see Figure 4(a)). Then we provide two sliders that the user can interactively tune to filter the curvature and select meaningful regions. These sliders represent:



(a) Mean curvature (b) Smoothed curvature

(c) Curvature thresholding (d) Topological closure

Figure 4: *Curvature analysis-based feature detection.*

- A number of iterations β of Laplacian diffusion on the curvature values. We define the i-th iteration of the Laplacian curvature diffusion as:

$$curv^{i+1}(n, M^t) = curv^i(n, M^t) - \lambda.\Delta_{curv^i}(n, M^t) \quad (2)$$

with $curv^0(n, M^t) = curv(n, M^t)$ and $i \in [0, \beta]$. In our experiment, we used $\lambda = 1$ as a diffusion factor. Laplacian diffusion of the computed curvature values is used to decrease the spatial frequency of the curvature function over the surface. This allows the user to select broader regions in an efficient way without actually smoothing the geometric details on the mesh (see Figure 4(b)).

- A threshold γ on the curvature of R. All the vertices whose curvature is above γ are added to R. This allows the user to control the extent of R (see Figure 4(c)).

In the end, we can mathematically define a region of interest for a frame t as:

$$R^t = \{n \in N_{M^t} | curv^\beta(n, M^t) > \gamma\} \quad (3)$$

Topological filtering. In many cases, curvature-based selection is not sufficient to extract meaningful animation features. For instance, in Figure 4(c), the user might want to select the whole crown splash and not only its contour as it has been done with the curvature analysis tool. To remedy these issues, we extend mathematical morphological operators (MMOs) to polygonal meshes. They allow the user to interactively and easily refine the regions of interest detected by the curvature analysis.

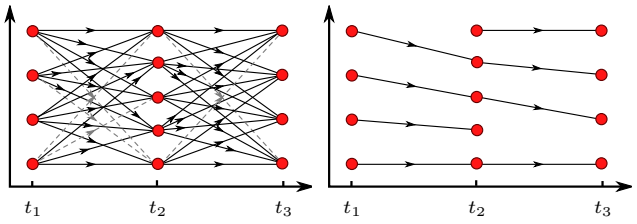
We propose two main tools:

- *Erosion* for disconnecting, reducing or removing parts of R.
- *Dilatation* for connecting and enlarging parts of R.

Both tools can be combined for performing *openings* and *closures* of R. In practice, these tools were particularly useful for selecting regions such as the interior part of the circular wave in Figure 4(d), achieved with a closure. For a detail overview of MMOs, we refer the reader to the work of Serra [Serra 1986] and to Appendix B.

4.2 Segmentation

Once R has been computed, the segmentation step decomposes each R^t into connected components $(C_k^t)_{k,t}$, where k is the index



(a) Initial DAG (b) Vertex-disjoint path cover

Figure 5: *Left: Frame features (red dots) are assembled into a directed acyclic graph as described in Section 4.3. Each edge of the graph carries a cost computed with Equation (5). Edges whose cost is over a user-defined threshold (gray dashed lines) are discarded. Right: a vertex-disjoint minimum-cost path cover has been computed based on Algorithm (1). The extracted paths represent space-time features.*

of the component. Mathematically, a region of interest for a frame t can be defined as the disjoint union of its connected components:

$$R^t = \bigsqcup_k C_k^t \mid \forall t \in [0, T(M) - 1] \quad (4)$$

The decomposition is computed using the straightforward breadth-first search on each frame in parallel. We call each C_k^t a *frame feature*.

4.3 Aggregation

Finally, the aggregation step extracts temporally coherent sequences of *frame features* that we call *space-time features*. The process is divided into two steps as illustrated in Figure 5. First, we build a graph of all possible *frame feature* connections, and then we compute a vertex-disjoint path cover of that graph. Temporal coherency of the resulting paths is enforced by minimizing a geometric matching cost described below. We call the resulting paths *space-time features*.

Graph construction. We build a directed acyclic graph $G = (V_G, E_G)$ representing the possible connections between frame features (see figure 5(a)). The set of nodes V_G is made of the frame features $\left(C_k^t\right)_{k,t}$ while the set of edges E_G is made of oriented edges e_{ij} linking each pair of consecutive frame features C_i^t and C_j^{t+1}.

Edge cost computation For every edge $e_{ij} \in E_G$, we compute a cost measure ω_{ij}. This measure relates to the geometrical matching between its two endpoints v_i and v_j. We divided ω_{ij} into three terms:

- d_{ij}: The distance between the centers of mass of v_i and v_j.
- s_{ij}: The difference of the surface area between v_i and v_j.
- v_{ij}: The difference of volume between v_i and v_j. v_{ij} is computed only if both v_i and v_j are closed.

The edge cost ω_{ij} is a weighted sum of these terms, normalized by the appropriate power of $l = L(M)$, the characteristic size of the bounding box of M:

$$\omega_{ij} = \omega_d \left(\frac{d_{ij}}{l}\right)^2 + \omega_s \left(\frac{s_{ij}}{l^2}\right)^2 + \omega_v \left(\frac{v_{ij}}{l^3}\right)^2 \quad (5)$$

For all the examples of this paper we used $(\omega_d, \omega_s, \omega_v) = (0.6, 0.2, 0.2)$. We chose to favour the closeness between frame features and consider difference of surface and volume equally. After the cost computation, we discard edges whose cost is above a threshold ϵ that we set to $0.3 \times l$ in our examples. Higher thresholds lead to fewer edges in the graph and more disconnected paths.

Vertex-disjoint path cover computation. To the authors' knowledge, there is no standard algorithm for computing minimum weight vertex-disjoint path cover. We propose an algorithm based on Kruskal's [1956] algorithm for computing minimum spanning trees: All vertices are first copied from the input graph to the output one; edges of the input graph are considered in ascending order of cost and added to the output graph if they satisfy a given topological condition. In Kruskal's algorithm, the condition is that the edge does not form a cycle in the output graph. In ours, the condition is that both of its endpoint vertices have strictly fewer than two neighbors. This allows us to ensure that the resulting path cover will be vertex-disjoint.

We detail our vertex-disjoint path cover process in Algorithm 1 using the following notation:

- G, V, and E represents respectively a graph, a set of vertices, and a set of edges;
- in and out subscripts refer to input and output elements;
- v_0^e and v_1^e refer to the endpoints of edge e in both G_{in} and G_{out} (since $V_{in} = V_{out}$);
- $deg(v)$ is the degree of vertex v in G_{out};
- $sort(E)$ is the in-place sort of the edges of E in the ascending cost order.

Appendix C contains the proof that Algorithm 1 produces an optimal solution to the minimal vertex disjoint path cover computation problem.

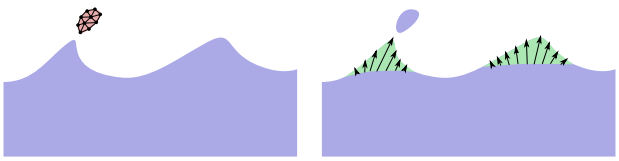
Algorithm 1 Vertex-disjoint path cover computation

$G_{in} = (V_{in}, E_{in})$
$G_{out} = (V_{out}, E_{out})$
$V_{out} = V_{in}$
$E_{out} = \varnothing$
$sort(E_{in})$
for all $e \in E_{in}$ **do**
  if $deg(v_0^e) < 2$ **and** $deg(v_1^e) < 2$ **then**
    $E_{out} \leftarrow e$
  end if
end for

At the end of this algorithm, the graph G_{out} consists of all frame-features V_{out} connected by inter-frame links E_{out}. E_{out} represents independent paths, as illustrated in Figure 5(b), which are optimal in the sense that the algorithm greedily minimizes our edge cost metric. These paths describe the *space-time features*.

5 Feature representation

Space-time features can be seen as a simple set of vertices belonging to M. This representation is, however, inconvenient for direct manipulation as it strongly depends on the input animation and therefore cannot be transferred from one animation to another. To be able to copy, edit and paste space-time features in different animations, we propose to build a representation of a space-time feature which is independent from M.



(a) Mesh representation (b) Differential representation

Figure 7: *Depending on whether the frame feature has closed boundaries or not, it is stored either as a mesh (left) or as a displacement field (right).*

In the remainder of the paper, we will note a space-time feature representation $F_i = \left(F_i^t\right)_{t^s(F_i) \le t \le t^e(F_i)}$ where $t^{s/e}(F_i)$ are the starting/ending frame index of F_i and F_i^t is the frame feature representation of F_i at the frame t. Also, we denote by $S(F_i^t)$ the mesh part of M^t corresponding to F_i^t.

We distinguish two representations depending on whether the frame feature has boundaries or not (see Figure 7). In the first case, we use a *mesh representation*, noted $M(F_i^t)$ and composed of a simple 3D mesh. It is used to represent a connected component of the liquid, such as droplet or a larger body of water. In the second case, we use a *differential representation*, noted $(\tau_d(F_i^t), \tau_n(F_i^t))$, and composed of a pair of textures representing a displacement map and a normal map. The displacement map is used to store the deformation of the feature and the normal map is used for aligning the feature to the surface. It is used for frame features representing a local sub-part of a larger body of water, such as a single wave on the surface of an ocean.

A space-time feature can be composed of frame features from both categories. A typical case of mixed representation is an isolated drop falling into a larger body of water and becoming a detail of this larger surface.

In the following of this section, we detail the computation of both representations and how they can be inserted back into a different animation. This will be useful later for copying and pasting features.

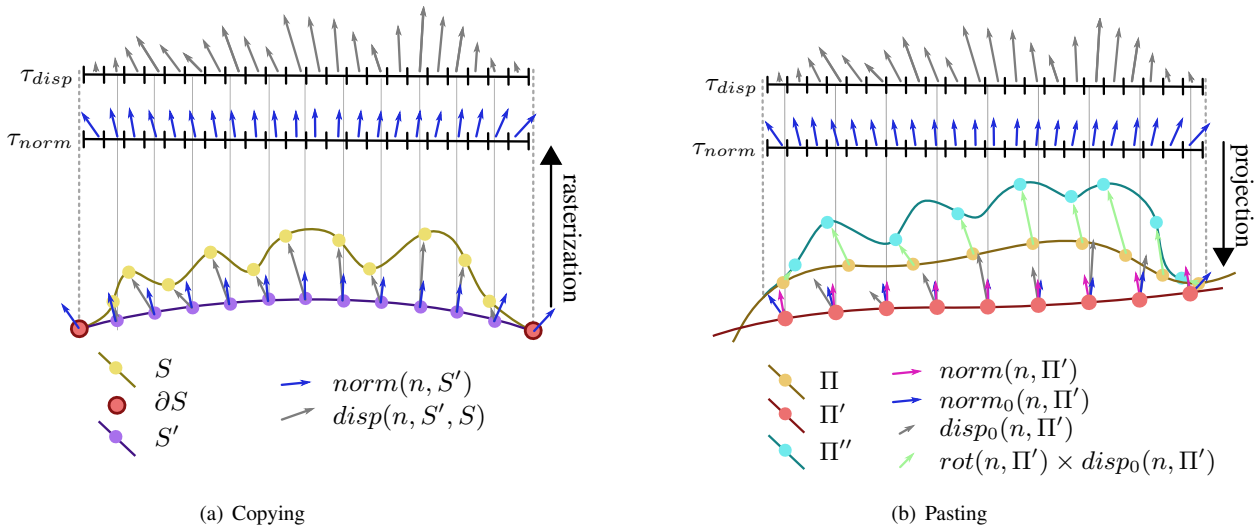
5.1 Computation

Building the mesh representation of a frame feature F_i^t simply consists of transforming $S(F_i^t)$ into an independent mesh $M(F_i^t)$.

Building the differential representation of a frame feature is slightly more complex. The process is described in Figure 6(a)), and consists of three steps: Starting from the initial frame feature surface $S(F_i^t)$ we compute a smooth version $S'(F_i^t)$ using single step Laplacian canceling on the inner part of the surface $S(F_i^t) \setminus \partial S(F_i^t)$. We note $pos(n, S)$ the position of vertex n on surface S; note that $S(F_i^t)$ and $S'(F_i^t)$ describes the same vertices, but with different positions. Then we compute the displacement of each vertex $n \in N$ from $S'(F_i^t)$ to $S(F_i^t)$:

$$disp(n, S'(F_i^t), S(F_i^t)) = pos(n, S(F_i^t)) - pos(n, S'(F_i^t)) \quad (6)$$

Finally, we map for every vertex n, $disp(n, S'(F_i^t), S(F_i^t))$ onto $S'(F_i^t)$, and sample the linearly interpolated values into the texture $\tau_d(F_i^t)$. We similarly sample the normals of $S'(F_i^t)$ into $\tau_n(F_i^t)$. The samplings are performed on the GPU using the standard off-screen rasterization pipeline. In order to avoid grid artifacts the resolution respect the Nyquist-Shannon theorem, i.e. allow each edge of the triangulation to be sampled by at least two pixels. In practice, we computed the resolution based on the bounding box of



(a) Copying

(b) Pasting

Figure 6: *Left: The displacement representation of a mesh part S is built from the sampling of the displacement field transporting S' toward S, and the normal field of S'. Right: This representation can be inserted back into a mesh part Π by projecting a displacement and a normal on vertices of Π'. The difference of normals between S' and Π' is used for orienting the displacements, which are in turn used for generating the deformed surface Π''.*

the frame feature that we normalized and multiplied by a resolution factor that we set to 1000. This induces artifacts with extremely small triangles; this point is discussed in Section 8 (§Resolution issues).

5.2 Insertion

Mesh representations are trivially inserted by copying $M(F_i^{t'})$ into M^t.

To insert a feature representation $(\tau_d(F_i^{t'}), \tau_n(F_i^{t'}))$ into M^t at location p, we first need to identify the part $\Pi \subset M^t$ which will be deformed. Starting from $N_\Pi = \{n_0\}$ where

$$n_0 = \underset{n \in N_{M^t}}{\operatorname{argmin}}(\|pos(n, M^t) - p\|) \tag{7}$$

we progressively dilate Π until it fills the bounding box of size $L(F_i^{t'})$ centered in p.

Once Π has been computed, we compute its smooth version Π' on which we project $\tau_d(F_i^{t'})$ and $\tau_n(F_i^{t'})$, yielding two attributes for each vertex $n \in N_{\Pi'}$, a displacement $disp_0(n, \Pi')$ and a normal $norm_0(n, \Pi')$. We define a new attribute $rot(n, \Pi') = rot(norm_0(n, \Pi'), norm(n, \Pi'))$, a rotation matrix mapping $norm_0(n, \Pi')$ into $norm(n, \Pi')$. Each vertex $n \in N_\Pi$ is displaced of $rot(n, \Pi') \times disp_0(n, \Pi')$, yielding the deformed surface Π''. These operations allow to counter the effects of low-resolution shapes of both $S(F_i^t)$ and Π. Figure 6(b) illustrates these steps.

6 Sculpting Tools

Once space-time features representations have been computed, they can either be manipulated by the user to modify the current liquid animation, or they can be extracted and re-used in another liquid animation to enrich it. This section described the set of tools we propose; they are essentially the space-time analogue of common tools used for sculpting static geometry [Ferley et al. 2000; Schmidt and Singh 2010; Takayama et al. 2011].

Selection The first thing one might want from an interaction system is to specify which of the multiple entities of the scene are to be interacted with. This is usually performed through object selection. In our case, objects are space-time features, and they can be selected and grouped by clicking on their shape at a given frame.

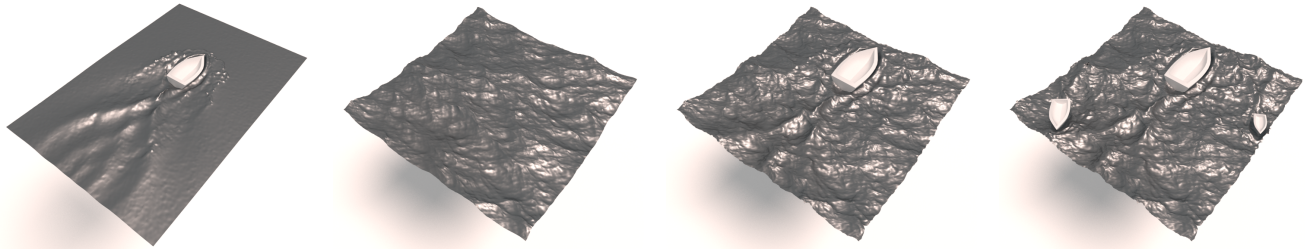
Copy and cut The copy operation consists of creating the representation of the selected features, as explained in Section 5.1. The cut operation is similar to the copy operation, except that the representation of the feature is removed from the animation after it has been computed. Once a feature or a feature group has been copied or cut, its representation becomes the current input data of further tools. It is later designated as "the current feature."

Export and import The current feature can be exported into a dedicated binary file format which stores its representation at each frame. This allows it to be imported back later to the same animation, or into a different one. Once imported, a feature becomes the current feature.

Paste The pasting operation allows a user to insert the current feature into a target animation, as explained in Section 5.2.

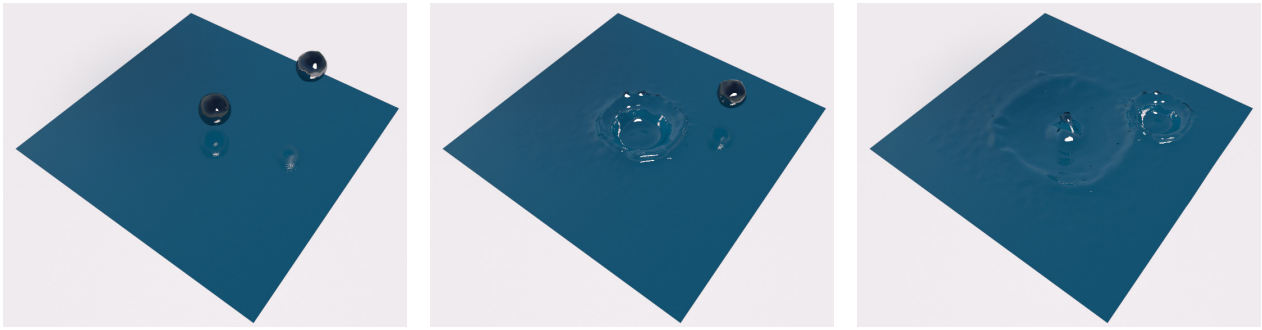
Space-time Deformation The user might want to use the feature in a different spatial and temporal configuration from the one in which it was extracted, so we propose adapted deformation tools. The position, orientation, and spatial scale of the current feature can be controlled with the mouse, and a real-time visual feed-back allows the user to set the feature in the configuration they require. By navigating in the animation, the user can also choose the initial frame of the current feature and set a time scale. This leads to a speed-up or slowdown of the feature animation.

Fade in and out When pasting a wave, the user can specify a fade in and a fade out interval. This means that that the feature will not immediately appear, but instead it smoothly grows in the beginning of its lifetime and smoothly disappears before the last



(a) Boat simulation (b) Procedural ocean (c) Pasting of the wake (d) Multiple pasting

Figure 8: *From the animation of boat generated using a FLIP simulation (left), our sculpting system allows to extract the wake of the boat in a single* space-time feature*. Then we can manipulate this feature and paste it into an ocean animation generated procedurally (middle). Editing the feature and pasting it multiple times allows to interactively model a complex scene (right) without re-simulating.*



(a) $t = t_0$ (b) $t = t_1$ (c) $t = t_2$

Figure 9: *From an existing liquid animation we extracted a complete crown splash into a single* space-time feature*. The feature combines both the fall of a drop and the resulting splash. We edit and paste this feature twice at different locations and modify the height of the droplets. Here, we show different frames of the final animation.*

frame of its lifetime. We achieve this effect by linearly blending the pasted displacement field over time with weights varying between 0 and 1.

Trajectory editing Space-time deformations influence all frames at once, whereas the user might want to control each frame individually. Per-frame spatial feature manipulation is achieved through a dedicated feature trajectory edit tool. This tool allows a user to displace the representation of a feature at a given frame while visualizing the positions of the feature at all the frames.

7 Results

In this section, we detail results achieved using our sculpting system. They illustrate the different tools described in the Section 6 and alternative usage of our method that we found interesting.

Boat wake In Figure 8, we illustrate the capability of our method to extract *space-time features* from arbitrary inputs (e.g. Eulerian or Lagrangian simulation, spectral methods, shallow water, real liquid surface acquisition) and combine them to create a plausible animation. We start from two animations: The first one (Figure 8(a)) was computed using the FLIP simulation method [Zhu and Bridson 2005] and represents a boat traversing a fluid tank and forming a wake. The second one (Figure 8(b)) is a procedural animation of ocean computed using the method of [Tessendorf 2004] and exhibits numerous small scale details. Then we extract the boat wake and paste it on the ocean animation at three different positions with different scales and orientations (Figures 8(c) and 8(d)).

Trajectory editing In Figure 9, we applied several edits to a *space-time feature* capturing a crown splash. First, we temporally remapped the feature to slow it down. Second, we pasted it twice on a static plane at different locations. Third, we edited the trajectory of the droplets to change the height of their fall. Finally, we used a fading out to obtain a smooth transition with the initial plane.

Animation enrichment An interesting aspect of our method is that it can be used to enrich static objects or non-fluid objects with a fluid-like behavior. In Figure 1, we enriched a static object with a splash extracted from a liquid animation. More generally, our method allows to combines results obtained with very different methods such as procedural animation, eulerian and lagrangian simulations, shallow water simulation, or artist-created animations.

8 Discussion

Our method is not without limitations, and we suggest several directions for future work.

Physical consistency Even though the space-time features selected by the user capture realistic behavior, the way they are edited and inserted may spoil the realism of the resulting animation. As we do not check for physical consistency, the plausibility of the result depends on the user's artistic skill. An extension of our method

would be to adapt the destination surface so that it matches the input features under physical constraints such as volume preservation. To incorporate further physical constraints such as momentum conservation, using mesh sequences as input would not be sufficient anymore and additional information such as velocity would be required. Designing an interactive editing method given these constraints may be difficult to achieve.

Resolution issues Geometrical details may be lost when copying because of the resolution of the stamp compared to the resolution of the triangulation. This problem is especially important with highly protruding features, and could be solved by using a vectorial representation for stamps.

Geometrical details may also be lost when pasting a feature if the resolution of the target mesh sequence is too coarse compared to that of the stamp. To remedy this limitation, we could add an automatic mesh refinement scheme such that the resolution of the target mesh always locally matches the resolution of details in the pasted feature.

Aggregation robustness The aggregation of regions of interest into space-time features is a key component of our approach. However, as it is based on geometrical similarities between two consecutive frames, it might fail if the time step between two frames is too large or if parts of the water body are moving too fast, such as in the case of dynamic splashes with lots of fast moving droplets. Even if it has not been an issue for the results of this paper, we would like to enforce the robustness of the aggregation step by adding a new metric which would measure the physical coherency between two regions of interest. This metric would take into account some inferred velocity for the region. It could also incorporate some cause and effect relationships; for example, a falling drop will cause waves.

Memory consumption For our results, we worked with short sequence of liquid animations but when editing a large sequence of high resolution meshes and extracting potentially large space-time features, memory consumption may become a problem. A classical solution would be to use a multi-resolution approach. The user would manipulate a low resolution version of the animation which would ensure interactivity. Then, the user's edits choice would be transferred to the high resolution version of the animation as an offline post-process.

Feature editing We proposed basic tools for the space-time edition of features and there are several avenues for future work. Firstly, our copy/paste method is only able to deal with simple deformations of a surface. By using the work of [Takahashi et al. 2003] to extract and insert displacement fields, we could handle much more complex cases. Secondly, we would like to propose a space-time sculpting tool close to space deformers such as constant volume tools [Angelidis et al. 2006; von Funck et al. 2006] and topology modifiers [Stanculescu et al. 2011]. The idea would be to let the user sculpt a specific frame and to interpolate the deformation over time, similar to what is proposed by Stuyck and Dutré [Stuyck and Dutré 2016] but without re-simulation.

Finally, we think it would be useful to let any edited parameter (scale, rotation, etc.) to be key-framed in order to make time-varying effects more easily controllable.

9 Conclusion

This paper introduces the first method for interactively editing existing fluid animations. Our method is based on an intuitive sculpting metaphor where the user can select, copy, edit and paste coherent *space-time features*. This approach allows a user to quickly design new liquid animations. In the future, we think that our representation for *space-time features* could be extended and used to manipulate animations at a higher level, similarly to a story-board.

10 Acknowledgements

This work was partly supported by the starting grant BigSplash, as well as the advanced grant EXPRESSIVE from the European Research Council (ERC-2014-StG_638176 , and ERC-2011-ADG_20110209).

References

ANGELIDIS, A., CANI, M.-P., WYVILL, G., AND KING, S. 2006. Swirling-sweepers: Constant-volume modeling. *Graph. Models 68*, 4 (July), 324–332.

BARBIČ, J., DA SILVA, M., AND POPOVIĆ, J. 2009. Deformable object animation using reduced optimal control. *ACM Trans. Graph. 28*, 3 (July), 53:1–53:9.

BARBIČ, J., SIN, F., AND GRINSPUN, E. 2012. Interactive editing of deformable simulations. *ACM Trans. Graph. 31*, 4 (July), 70:1–70:8.

BOTSCH, M., PAULY, M., KOBBELT, L., ALLIEZ, P., LÉVY, B., BISCHOFF, S., AND RÖSSL, C. 2007. Geometric modeling based on polygonal meshes video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 Courses*, ACM, New York, NY, USA, SIGGRAPH '07.

CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 219–228.

FATTAL, R., AND LISCHINSKI, D. 2004. Target-driven smoke animation. *ACM Trans. Graph. 23*, 3 (Aug.), 441–448.

FERLEY, E., CANI, M.-P., AND GASCUEL, J.-D. 2000. Practical Volumetric Sculpting. *Visual Computer 16*, 8, 469–480.

HINSINGER, D., NEYRET, F., AND CANI, M.-P. 2002. Interactive Animation of Ocean Waves. In *ACM-SIGGRAPH - EG Symposium on Computer Animation (SCA'02)*.

HONG, J.-M., AND KIM, C.-H. 2004. Controlling fluid animation with geometric potential: Research articles. *Comput. Animat. Virtual Worlds 15*, 3-4 (July), 147–157.

HORVATH, C. J. 2015. Empirical directional wave spectra for computer graphics. In *Proceedings of the 2015 Symposium on Digital Production*, ACM, 29–39.

HUANG, R., AND KEYSER, J. 2013. Automated sampling and control of gaseous simulations. *The Visual Computer 29*, 6-8, 751–760.

IHMSEN, M., AKINCI, N., AKINCI, G., AND TESCHNER, M. 2012. Unified spray, foam and air bubbles for particle-based fluids. *Vis. Comput. 28*, 6-8 (June), 669–677.

JESCHKE, S., AND WOJTAN, C. 2015. Water wave animation via wavefront parameter interpolation. *ACM Transactions on Graphics (TOG) 34*, 3, 27.

KIM, Y., MACHIRAJU, R., AND DAVID, T. 2006. Path-based control of smoke simulations. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '06, 33–42.

KRUSKAL, J. B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society 7*, 1, 48–50.

LI, S., HUANG, J., DE GOES, F., JIN, X., BAO, H., AND DESBRUN, M. 2014. Space-time editing of elastic motion through material optimization and reduction. *ACM Trans. Graph. 33*, 4 (July), 108:1–108:10.

MADILL, J., AND MOULD, D. 2013. Target particle control of smoke simulation. In *Proceedings of Graphics Interface 2013*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, GI '13, 125–132.

MCNAMARA, A., TREUILLE, A., POPOVIĆ, Z., AND STAM, J. 2004. Fluid control using the adjoint method. *ACM Trans. Graph. 23*, 3 (Aug.), 449–456.

NARAIN, R., KWATRA, V., LEE, H.-P., KIM, T., CARLSON, M., AND LIN, M. C. 2007. Feature-guided dynamic texture synthesis on continuous flows. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, EGSR'07, 361–370.

NIELSEN, M. B., AND BRIDSON, R. 2011. Guide shapes for high resolution naturalistic liquid simulation. *ACM Trans. Graph. 30*, 4 (July), 83:1–83:8.

NIELSEN, M. B., AND CHRISTENSEN, B. B. 2010. Improved variational guiding of smoke animations. *Computer Graphics Forum 29*, 2, 705–712.

NIELSEN, M. B., CHRISTENSEN, B. B., ZAFAR, N. B., ROBLE, D., AND MUSETH, K. 2009. Guiding of smoke animations through variational coupling of simulations at different resolutions. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '09, 217–226.

PAN, Z., HUANG, J., TONG, Y., ZHENG, C., AND BAO, H. 2013. Interactive localized liquid motion editing. *ACM Trans. Graph. 32*, 6 (Nov.), 184:1–184:10.

PIGHIN, F., COHEN, J. M., AND SHAH, M. 2004. Modeling and editing flows using advected radial basis functions. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '04, 223–232.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. 2000. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 209–217.

RAVEENDRAN, K., THUEREY, N., WOJTAN, C., AND TURK, G. 2012. Controlling liquids using meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '12, 255–264.

RAVEENDRAN, K., WOJTAN, C., THUEREY, N., AND TURK, G. 2014. Blending liquids. *ACM Trans. Graph. 33*, 4 (July), 137:1–137:10.

REISCH, J., MARSHALL, S., WRENNINGE, M., GÖKTEKIN, T., HALL, M., O'BRIEN, M., JOHNSTON, J., REMPEL, J., AND LIN, A. 2016. Simulating rivers in the good dinosaur. In *ACM SIGGRAPH 2016 Talks*, ACM, New York, NY, USA, SIGGRAPH '16, 40:1–40:1.

SCHMIDT, R., AND SINGH, K. 2010. Meshmixer: an interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*, ACM.

SCHPOK, J., DWYER, W., AND EBERT, D. S. 2005. Modeling and animating gases with simulation features. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '05, 97–105.

SCHULZ, C., VON TYCOWICZ, C., SEIDEL, H.-P., AND HILDEBRANDT, K. 2014. Animating deformable objects using sparse spacetime constraints. *ACM Trans. Graph. 33*, 4 (July), 109:1–109:10.

SERRA, J. 1986. Introduction to mathematical morphology. *Computer vision, graphics, and image processing 35*, 3, 283–305.

SHI, L., AND YU, Y. 2005. Controllable smoke animation with guiding objects. *ACM Trans. Graph. 24*, 1 (Jan.), 140–164.

SHI, L., AND YU, Y. 2005. Taming liquids for rapidly changing targets. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '05, 229–236.

STANCULESCU, L., CHAINE, R., AND CANI, M.-P. 2011. Freestyle: Sculpting meshes with self-adaptive topology. In *Computers & Graphics*.

STUYCK, T., AND DUTRÉ, P., 2016. Sculpting fluids: A new approach to art-directable fluids. SIGGRAPH 2016 Poster, SIGGRAPH 2016, Anaheim, California, USA, 1 page, 24-28 July 2016., July.

TAKAHASHI, T., FUJII, H., KUNIMATSU, A., HIWADA, K., SAITO, T., TANAKA, K., AND UEKI, H. 2003. Realistic animation of fluid with splash and foam. *Computer Graphics Forum 22*, 3, 391–400.

TAKAYAMA, K., SCHMIDT, R., SINGH, K., IGARASHI, T., BOUBEKEUR, T., AND SORKINE, O. 2011. Geobrush: Interactive mesh geometry cloning. *Computer Graphics Forum (proceedings of EUROGRAPHICS) 30*, 2, 613–622.

TESSENDORF, J. 2004. Simulating ocean surface. In *Siggraph course notes*, ACM.

THÜREY, N., KEISER, R., PAULY, M., AND RÜDE, U. 2006. Detail-preserving fluid control. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '06, 7–12.

TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Trans. Graph. 22*, 3 (July), 716–723.

TWIGG, C. D., AND JAMES, D. L. 2007. Many-worlds browsing for control of multibody dynamics. *ACM Trans. Graph. 26*, 3 (July).

VAN OPSTAL, B., JANIN, L., MUSETH, K., AND ALDÉN, M. 2014. Large scale simulation and surfacing of water and ice ef-

fects in dragons 2. In *ACM SIGGRAPH 2014 Talks*, ACM, New York, NY, USA, SIGGRAPH '14, 11:1–11:1.

VON FUNCK, W., THEISEL, H., AND SEIDEL, H.-P. 2006. Vector field based shape deformations. In *ACM TOG, proc. of SIGGRAPH*.

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '88, 159–168.

WOJTAN, C., MUCHA, P. J., AND TURK, G. 2006. Keyframe control of complex particle systems using the adjoint method. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, 15–23.

YANG, B., LIU, Y., YOU, L., AND JIN, X. 2013. Technical section: A unified smoke control method based on signed distance field. *Comput. Graph. 37*, 7 (Nov.), 775–786.

YUAN, Z., CHEN, F., AND ZHAO, Y. 2011. Pattern-guided smoke animation with lagrangian coherent structure. *ACM Trans. Graph. 30*, 6 (Dec.), 136:1–136:8.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. In *ACM Transactions on Graphics (TOG)*, vol. 24, ACM, 965–972.

A Notation table

Symbol	Description
$M = (M^t)_{0 \leq t < T(M)}$	Mesh sequence
$T(.)$	Number of frames in argument
$L(.)$	Characteristic length in argument
N_X	Nodes of mesh X
P_X	Polygonal faces of mesh X
$pos(n, X)$	Position of vertex $n \in N_X$
$norm(n, X)$	Normal of vertex $n \in N_X$
$disp(n, X_1, X_2)$	$pos(n, X_2) - pos(n, X_1)$
$F_i = (F_i^t)_{t^s(F_i) \leq t \leq t^e(F_i)}$	i^{th} feature of the animation
$t^{s/e}(F_i)$	Starting/ending frame index of F_i
$S(F_i^t)$	Part of M^t corresponding to F_i^t
X'	Smoothed version of mesh or mesh part X
$M(F_i^t)$	Mesh representation of F_i^t
$(\tau_d(F_i^t), \tau_n(F_i^t))$	Differential representation of F_i^t
$C(X)$	Center of mass of surface X
$A(X)$	Area of surface X
$V(X)$	Volume of surface X
G	Frame features adjacency graph
$V_G = \{F_j^t\}_{t,j}$	Vertices of graph G
$E_G = \{e_{ij}^t\}_{t, r(i,j)}$	Edges of graph G
ω_{ij}^t	Cost of e_{ij}^t (see Equation 5)
d_{ij}^t	$\|C(S(F_i^t)) - C(S(_j^{t+1}))\|$
a_{ij}^t	$\|A(S(F_i^t)) - A(S(_j^{t+1}))\|$
v_{ij}^t	$\|V(S(F_i^t)) - V(S(_j^{t+1}))\|$
ω_d	Edge cost distance weight
ω_a	Edge cost area weight
ω_v	Edge cost volume weight

Table 1: *Notations used throughout this article. Subscript (resp. superscript) indices are used for spatial (resp. temporal) indexing. Parenthesis (resp curly braces) are used for ordered (resp unordered) sets.*

B Mesh part manipulation

We define R a part of mesh $X = (N_X, P_X)$ as a subset of its vertices: $R \subset N_X$. R being itself a set, usual set operations can be performed on it such as union, intersection and difference.

The border of R, noted ∂R, is defined by:

$$\partial R = \{n \in R | \exists n_i \in neib(n), n_i \notin R\} \tag{8}$$

where $neib(n)$ is the set of neighbors of n in X:

$$neib(n) = \{n' \in N_X | \exists p \in P_X, n \in p \cup n' \in p\} \tag{9}$$

Mathematical morphology operators. R can be *eroded* into $ero(R)$ using the following relation:

$$ero(R) = \{n \in R | \nexists n_i \in neib(n), n_i \notin R\} \tag{10}$$

which is equivalent to $ero(R) = R \setminus \partial R$.

Reciprocally, R can be *dilated* into $dil(R)$ using the following relation:

$$dil(R) = \{n \in N_X | \exists n_i \in neib(n), n_i \in R\} \tag{11}$$

Dilation can be extended to an arbitrary order k:

$$dil^k(R) = \underbrace{dil(\dots dil(}_{k \text{ terms}} R) \dots) \tag{12}$$

The same stands for erosion:

$$ero^k(R) = \underbrace{ero(\dots ero(}_{k \text{ terms}} R) \dots) \tag{13}$$

We call *opening* of order k the mesh part defined as:

$$ope^k(R) = dil^k(ero^k(R)) \tag{14}$$

and *closure* of order k the mesh part defined as:

$$clo^k(R) = ero^k(dil^k(R)) \tag{15}$$

C Proof of correctness

This appendix contains the proof that Algorithm 1 produces optimal minimum vertex disjoint path covers. The proof consists of two parts: First, it is proved that the algorithm produces a vertex disjoint path cover; Second, it is proved that the constructed vertex disjoint path cover is of minimal weight.

Vertex disjoint path cover Let $\mathcal{G}$ be a connected, edge-weighted graph and let $\mathcal{Y}$ be the subgraph of $\mathcal{G}$ produced by the algorithm. For a vertex v of $\mathcal{Y}$ to belong to more than one path, it require two or more edges to be arriving to or departing from it. However, the algorithm allows at most one edge to depart from and to arrive to any vertex. As a result, $\mathcal{Y}$ is a vertex disjoint path cover.

Minimality We show that the following proposition $\mathcal{P}$ is true by induction: If $\mathcal{F}$ is the set of edges chosen at any stage of the algorithm, then there is some vertex disjoint path cover that contains $\mathcal{F}$.

- Clearly $\mathcal{P}$ is true at the beginning, when $\mathcal{F}$ is empty: Any vertex disjoint path cover will do, and there exists one because the subgraph composed only of the vertices of $\mathcal{G}$ and no edges is a minimum vertex disjoint path cover of $\mathcal{G}$.

- Now assume $\mathcal{P}$ is true for some non-final edge set $\mathcal{F}$ and let $\mathcal{T}$ be a vertex disjoint path cover that contains $\mathcal{F}$. If the next chosen edge e is also in $\mathcal{T}$, then $\mathcal{P}$ is true for $\mathcal{F} \cup \{e\}$. Otherwise, $\mathcal{T} \cup \{e\}$ has a vertex v belonging to exactly two paths. Let us call e' the edge of $\mathcal{T}$ connected to v with the same direction than e. Then $\mathcal{T} \cup \{e\} \setminus \{e'\}$ is a vertex disjoint path cover, and it has the same weight as $\mathcal{T}$, since $\mathcal{T}$ has minimum weight and the weight of e' cannot be less than the weight of e, otherwise the algorithm would have chosen e' instead of e. Therefore, $\mathcal{P}$ is true for $\mathcal{F} \cup \{e\}$ in any case.
- Therefore, by the principle of induction, $\mathcal{P}$ holds when $\mathcal{F}$ has become a vertex disjoint path cover, which is only possible if $\mathcal{F}$ is a vertex disjoint path cover itself.