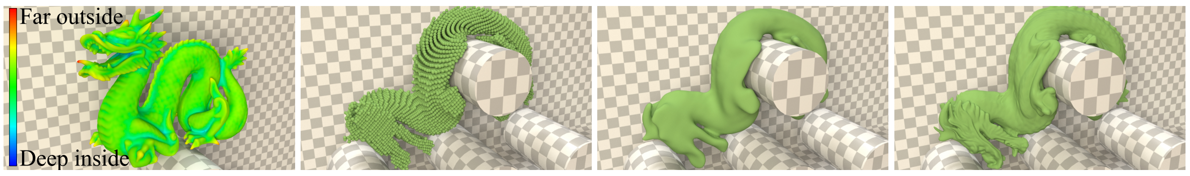
Detail-Preserving Explicit Mesh Projection and Topology Matching for Particle-Based Fluids

F. Dagenais[1,†] J. Gagnon[1,2,‡] E. Paquette[1,§]

[1]Multimedia Lab, École de technologie supérieure, Montreal, Canada
[2]Mokko Studio, Montreal, Canada



(a) (b) (c) (d)

Figure 1: High-resolution details are extracted based on the distance between an initial surface mesh and its coarse implicit surface representation (a). We use the particles' behavior and the topology of a coarse resolution simulation (b) to consistently evolve an explicit surface mesh (d). The high-resolution details are lost if using only the implicit surface from the coarse set of particles (c), but our explicit mesh surface approach preserves surface details (d), even with such a coarse resolution fluid simulation. In (a), the surface is color-coded based on the distance from the implicit surface, where blue refers to the largest inward distances, red refers to the largest outward distances, and green is at the isosurface. The coarse simulation (b) contains only 20,456 particles.

Abstract

We propose a new explicit surface tracking approach for particle-based fluid simulations. Our goal is to advect and update a highly detailed surface, while only computing a coarse simulation. Current explicit surface methods lose surface details when projecting on the isosurface of an implicit function built from particles. Our approach uses a detail-preserving projection, based on a signed distance field, to prevent the divergence of the explicit surface without losing its initial details. Furthermore, we introduce a novel topology matching stage that corrects the topology of the explicit surface based on the topology of an implicit function. To that end, we introduce an optimization approach to update our explicit mesh signed distance field before remeshing. Our approach is successfully used to preserve the surface details of melting and highly viscous objects, and shown to be stable by handling complex cases involving multiple topological changes. Compared to the computation of a high-resolution simulation, using our approach with a coarse fluid simulation significantly reduces the computation time and improves the quality of the resulting surface.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Particles are commonly used in fluid simulation, particularly with the rise in popularity of the FLIP method [ZB05] found in commercial software (such as Houdini™, Maya®, and

† e-mail: francois.dagenais.2@ens.etsmtl.ca
‡ e-mail: jonathangagnon@gmail.com
§ e-mail: eric.paquette@etsmtl.ca

Realflow^{TM}). When simulating liquids and reconstructing the surface from the particles, the irregular distribution of particles, as well as the spherical nature of the implicit functions typically used for reconstruction, make the resulting surface prone to bumpiness. This is shown in Fig. 2, where the Stanford Armadillo is reconstructed using different numbers of particles. As can be seen, a coarser simulation will severely smooth the details; therefore, a large amount of particles is needed to accurately reconstruct the details of the original mesh. Even in Fig. 2(d), where over four million particles were used, some details around the eyes and the teeth have been smoothed out. When melting objects using a highly viscous simulation, it is important to retain the details of the original mesh, and this requires a large number of particles. However, such a detailed simulation might be unnecessary, considering the small amount of detail present in the fluid movement. This makes the simulation and surface reconstruction times unnecessarily longer, whereas the objective is only to improve the reconstructed surface.

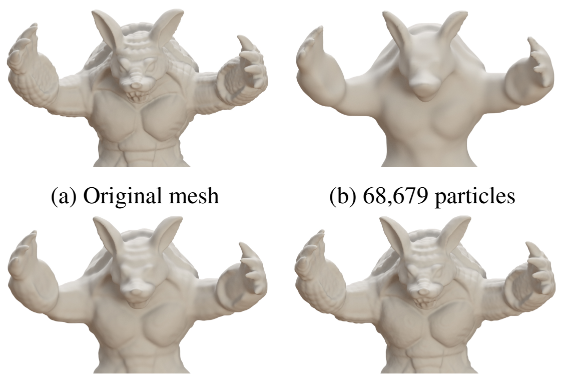

(a) Original mesh (b) 68,679 particles
(c) 549,485 particles (d) 4,395,521 particles

Figure 2: Comparison between the original surface and reconstructions with varying resolutions. The surfaces were generated using the method of Solenthaler et al. [SSP07].

Explicit surface tracking [WMFB11] helps solve this problem by using an initial mesh and evolving it based on the underlying simulation. Remeshing is done throughout the process in order to preserve the coherence and the quality of the surface. However, explicit surface tracking is not well suited for particle-based simulations since the explicit mesh tends to diverge from the particles. While recent work address this issue [YWTY12], they do not preserve surface details very well. Since they rely on a projection onto a surface constructed from the particles, the size of the explicit surface's details is still limited by the resolution of the underlying simulation.

Our approach focuses on improving and extending explicit surface tracking approaches for particle-based fluids in order to retain surface details while maintaining a behavior consistent with the simulation particles. It does so by extracting high-resolution details based on the distance between an initial surface mesh and a coarse implicit surface representation built from the particles (see Fig. 1). This distance is then preserved by our improved projection. Furthermore, we introduce a novel topology matching operation that preserves the consistency of the explicit surface with the behavior of the particles. Altogether, this allows the tracking of a detailed explicit mesh surface by using a coarser particle simulation. Moreover, the approach can be run entirely as a post-simulation step, and it is independent of the simulation approach, having been used with both FLIP and SPH simulations. Our main contributions can be summarized as follows:

- A new explicit mesh surface tracking technique for particle-based simulations.
- A detail-preserving projection based on a signed distance field.
- A novel topology matching approach that reconstructs a section of a mesh voxelization based on the topology of a reference signed distance field.

2. Related Work

A wide range of work was focused on tracking the surface of liquid simulations. This section describes these papers according to the quality of the surface, the amount of details of the surface, and whether they apply to particle-based simulations.

2.1. Implicit surface tracking

The most common fluid surfacing techniques build an implicit representation of the surface encoded into a grid-like structure. A triangle mesh is generated at each frame from the implicit representation using a method such as marching cubes [LC87].

A common way to represent the surface of a liquid with an Eulerian simulation involves using a level set. Foster and Fedkiw [FF01] introduced a level set method that uses particles, where both the level set and the particles are advected using the fluid velocity field. Particles are used to correct the level set after each advection stage, thus reducing volume loss. Later, this method was improved by Enright et al. [EMF02] with particles on both sides of the interface, further improving volume preservation. Some papers have adapted this method to support adaptive grids [LGF04] and grids with finer resolutions than the underlying simulation [KSK09], making it possible to create a more detailed surface using a coarse simulation. Even though these methods rely on particles at the interface, the particles are not used as surface representations.

While the previously mentioned methods provide a smooth surface, surface tracking has to be performed as the simulation is computed. Particle-based simulations, however, do not impose such requirements, allowing a more flexible workflow, where the surface can be parameterized and built after the desired liquid behavior is obtained. Müller et al. [MCG03] first extended the traditional *blobbies* to particle fluids by using the SPH framework to create a surface from liquid particles. Even though the surface field is smoothly interpolated between particles, the resulting surface is prone to bumpiness. Later, Zhu and Bridson [ZB05] developed a method based on the distance to the average position of particles within an influence radius. This method provides a smoother surface, but is prone to generating erroneous volumes of fluids between splashes and in concave regions. This problem was resolved by Solenthaler et al. [SSP07] with the use of an attenuation factor in those regions. Adams et al. [APKG07] also improved the technique of Zhu and Bridson by storing a signed distance field at each particle's position, which allowed the generation of a smoother surface with irregular particle distribution and with particles of varying radii. However, updating the sign distance field makes it more computationally intensive than previous approaches. Yu and Turk [YT13] used anisotropic kernels to provide a smooth surface. Their method generates a smooth surface while being faster than that of Adams et al. [APKG07]. However, it involves longer computation times than the method of Zhu and Bridson [ZB05], and suffers from volume shrinkage because the particles' positions are smoothed. In their adaptive liquid simulation framework, Ando et al. [ATW13] used the union of convex hulls built from groups of three particles to generate a smoother surface, at the expense of longer computation times. Other papers focus on first generating a lower quality surface, and smoothing it afterward. The smoothing operation is either done on the triangulated mesh [Wil08], or on the implicit function [BGB11].

All these methods [APKG07, ATW13, BGB11, MCG03, SSP07, Wil08, YT13, ZB05] generate a new surface from the simulation particles for every frame of the animation. Therefore, they depend on the resolution of the underlying simulation to provide a detailed surface. As such, they are not very accurate in preserving the features from the initial surface used to generate the simulation particles. This can be particularly problematic when a highly detailed surface is needed, while a coarse simulation can effectively capture the desired behavior. A good example would be the simulation of a melting object, where the initial surface is a detailed user-supplied mesh.

2.2. Explicit surface tracking

In order to preserve the details of the original surface, other methods update an explicit mesh surface throughout the simulation. Brochu and Bridson [BB09] developed a framework for tracking explicit surfaces where triangle collisions are computed, and mesh surgery is performed directly on the mesh triangles to handle splitting and merging of the surface. Mesh improvement operations, such as edge collapse and edge splitting, are also performed on the surface to guarantee its quality. Their framework has successfully been used to model the surface of liquids with thin features [EB14], and has been extended to handle multimaterial interfaces [DBG14]. Müller [Mül09] advects an explicit mesh using the velocity field of an Eulerian simulation. The mesh is then voxelized and rebuilt entirely. The explicit mesh allows the identification of cells containing thin sheets of liquid, which are then replicated during its reconstruction, instead of splitting the fluid. However, other small-scale surface details, such as ripples, are not preserved. Wojtan et al. [WTGT09] developed a method where merging and splitting regions are identified by comparing the mesh with its voxelization. These regions are then remeshed locally from the voxelization. This effectively preserves features of the explicit mesh in regions where no topological changes have occurred. Furthermore, the quality of the topology change detection and the remeshing is only dependent on the voxelization grid resolution. Thus, the simulation resolution can be coarser while preserving a detailed surface. The same authors [WTGT10] improved this work by maintaining sheets of liquids thinner than the grid resolution. Although the grid resolution still affects the quality of the surface, coarser grids can be used than with their previous work. These techniques have been used with Eulerian fluid simulations [Mül09, WTGT09, WTGT10], as well as with Lagrangian finite element methods (FEM) [WTGT09] and the discontinuous Galerkin method [EB14]. In all cases, the explicit mesh is used by the fluid simulation solver to track the fluid's interior.

Recently, Yu et al. [YWTY12] adapted the work of Wojtan et al. [WTGT09] for particle-based simulations. Because particles, instead of the explicit mesh, are used by the solver to track the fluid's interior, a divergence is often seen between the explicit surface and the particles during mesh advection. To prevent this problem, the mesh is projected onto an implicit surface representation every frame. While this projection stage does not preserve surface details, this is however not a problem in their case since their goal is to use an already coarse mesh to track surface properties over time. While we also project the mesh vertices on an isosurface of the particles' implicit function, the projection of Yu et al., based on the mesh vertices' normal, is less precise than ours, and does not preserve the explicit mesh surface details.

Explicit surface representations used with Eulerian or FEM simulations have the advantage of being able to preserve surface details without the need for a finer simulation resolution, but the fluid simulation and explicit surface advection are interlinked, forcing the simultaneous calculation of both. On the other hand, with particle-based simulations, it is possible to rebuild a new surface using modified parameters without having to run the simulation all over. However, to preserve the consistency of the explicit surface with the

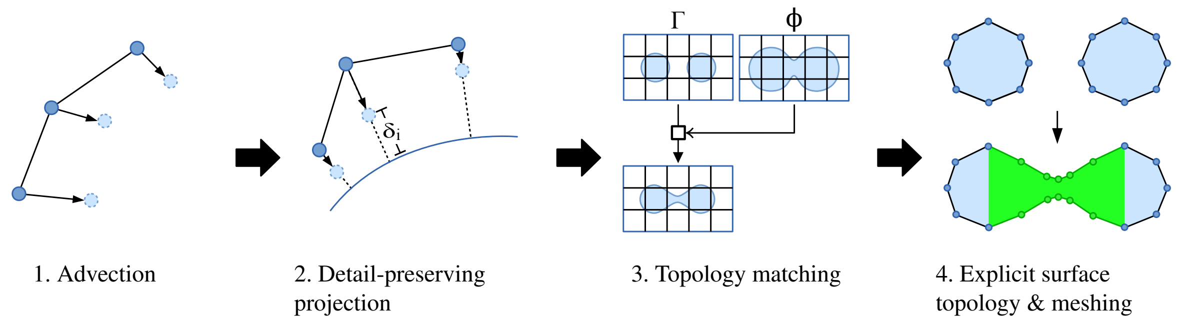

Γ
ϕ
δi
1. Advection
2. Detail-preserving projection
3. Topology matching
4. Explicit surface topology & meshing

Figure 3: Overview of our four-step approach.

simulation particles, current approaches project the explicit surface onto a coarser surface built from the particles, which then loses high-resolution surface details. The approach proposed in this paper focuses on preserving these surface details, while enforcing the consistency of the explicit surface with the simulation particles.

3. Overview

We introduce a new detail-preserving surface projection approach that extends the work of Yu et al. [YWTY12]. Using this projection, combined with a novel topology matching operation, changes in the topology of a coarse implicit surface are reflected onto a detailed explicit mesh surface. The global approach (see Fig. 3) consists of four main stages:

1. Advection [YWTY12] (Sec. 4)
2. New detail-preserving projection (Sec. 5)
3. New topology matching (Sec. 6)
4. Explicit surface topology & meshing [WTGT09] (Sec. 7)

During the first stage, the position of the explicit mesh vertices is updated using the underlying simulation position variation. Afterward, the explicit mesh is projected (stage 2) using the fluid implicit function, while preserving the mesh features. After the mesh has been advected and projected, a voxelization of the explicit mesh is computed and modified to match the topological changes of the implicit surface (stage 3). Finally, the modified mesh voxelization is used to locally rebuild the explicit mesh surface (stage 4) based on the topological changes (identified in stages 2 and 4).

3.1. Data structures

The approach presented in this paper maintains an explicit mesh structure and three scalar grid data structures. The three grid structures are the mesh voxelization Γ, the discretized implicit function ϕ, and the difference field ψ. A grid cell can be accessed either with its 1D index (e.g., ϕ_i) in memory, its 3D index (e.g., $\phi_{i,j,k}$) in space, or a vector (e.g., $\phi(\mathbf{x}_i)$) denoting a position in space. The grid resolutions are based on r, the initial distance between simulation particles. To preserve temporal coherence, the cell size is kept fixed, and the location of cells is kept fixed along the x, y, and z axes. To adapt to the changing fluid, only the extent of the grids is updated at each frame to fully surround the explicit mesh and the particles.

4. Explicit Mesh Advection

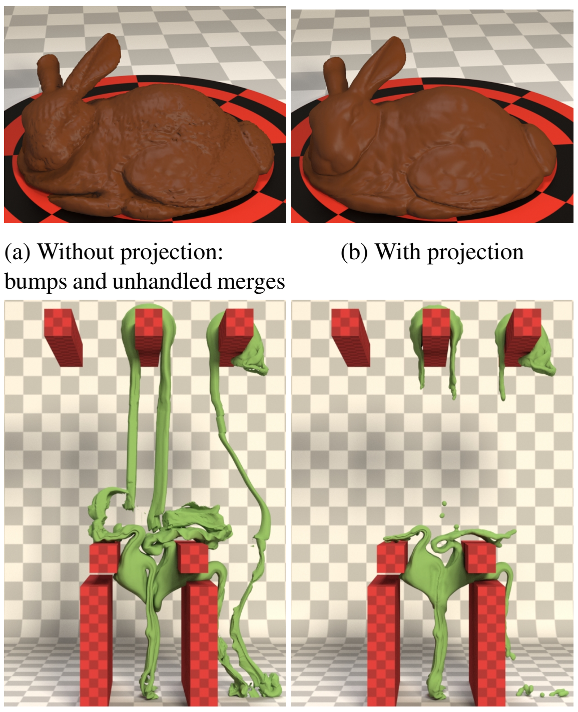
In order to advect the explicit mesh, the method of Yu et al. [YWTY12] uses the particles' velocities to update the vertices' positions at every timestep of the simulation. In this paper, the advection is done only at each frame, and relies on particle position updates. Using the position update instead of the particle velocity allows the surface to be advected only once per frame, instead of once per simulation timestep, thus reducing the computational expense. Furthermore, this eliminates the need to store particle data after every timestep in order to update the surface after the simulation is run. The particle position update $\Delta\mathbf{p}_j$ is used as follows:

$$\mathbf{x}_i^{t+\Delta t} = \mathbf{x}_i^t + \frac{\sum_j \Delta\mathbf{p}_j W_{ij}}{\sum_j W_{ij}},$$

where W_{ij} is the weighting function and $\mathbf{x}_i$ is the vertex position. In the examples shown in this paper, the *poly6* kernel from Müller et al. [MCG03] is used. The influence radius is set to $2r$, where r it the initial distance between the simulation particles. When a vertex has no neighbor within the influence radius, the radius is expanded until at least one particle is found. This guarantees that every vertex of the explicit mesh is updated.

5. Detail-Preserving Projection

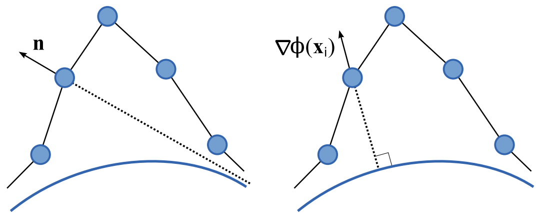
The advection stage described earlier is prone to small errors that accumulate during the course of the simulation. This results in a surface that diverges slowly from the particles, and in small bumps that tend to gradually appear at the surface of the mesh (see Fig. 4(a)). Furthermore, two surfaces moving toward each other do not always merge as they should. This problem is depicted in Fig. 4(a), where the front paws of the



(a) Without projection: bumps and unhandled merges (b) With projection

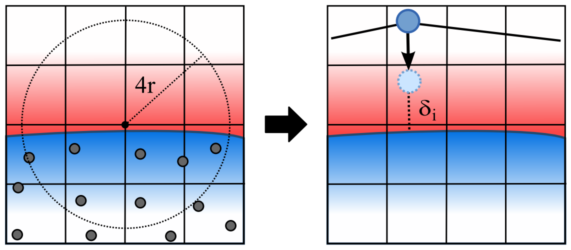
(c) Without projection: unrealistic stretching and volume gain (d) With projection

Figure 4: Examples without (left) and with (right) our detail-preserving projection.



(a) Vertex's normal (b) SDF

Figure 5: Projection using (a) the vertex's normal vs. (b) a signed distance field.



(a) Build SDF from particles (b) Project vertices

Figure 6: Detail-preserving projection steps. (a) First, a signed distance field is built from the particles; (b) then, it is used to project the vertices, while preserving the initial distance δ_i to the surface.

bunny did not merge with its chest, as would have been expected. This is also true with splitting (see Fig. 4(c)), and results in visually unpleasant stretching and volume gain. As in the work of Yu et al. [YWTY12], the explicit mesh is projected onto the isosurface of an implicit function. While Yu et al. projected along the vertex normals of the explicit mesh, we project using the gradient of the implicit function, which is a signed distance field. Projecting along this gradient is much more precise, as with our detailed surfaces, the normal might not be in the direction of the shortest path to the surface, and may not even cross the surface (see Fig. 5). Moreover, it might cross the projection path of another vertex, resulting in a surface with flipped triangles.

The projection is based solely on the signed distance field, and details are preserved by maintaining the same distance between a vertex of the explicit mesh and its projection for each frame of the animation. This is done in two steps (see Fig. 6):

1. Construction of the implicit function ϕ (Fig. 6(a) & Sec. 5.1)
2. Projection of the explicit mesh (Fig. 6(b) & Sec. 5.2)

5.1. Implicit function

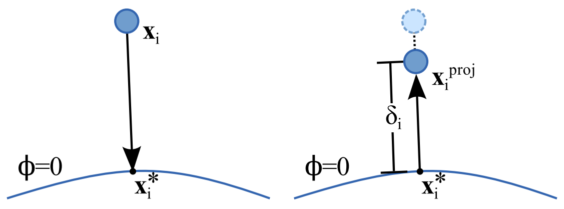
We choose our implicit function ϕ to be a signed distance field. This guarantees that, for any vertex's position $\mathbf{x}_i$, the gradient $\nabla\phi(\mathbf{x}_i)$ will be in the direction of the nearest point on the isosurface $\phi = 0$. Furthermore, the distance between that point and the mesh vertex is equal to $\phi(\mathbf{x}_i)$, which will be used during projection. To construct ϕ, the signed distance values around the isosurface $\phi = 0$ are first computed from the particles. Then, the distance information of interface cells is propagated further away by using the fast marching method of Sethian [Set95].

The mesh vertices are relatively close to the implicit surface, and computing the signed distance field is rather costly. So, we only compute it locally, based on the previous maximal distance between the mesh and the implicit surface. We thus use 1.25 times the previous maximal distance in our examples.

While several methods for surfacing particle fluids could be used to compute the values around the isosurface $\phi = 0$, the method described by Solenthaler et al. [SSP07] is used in this paper. In that method, an implicit function value is computed at each grid vertex position $\mathbf{x}_i$ using the average position of the neighbor particles inside an influence radius.

In order to prevent erroneous results between distant particles, a correction is also applied based on the maximum eigenvalue of the average position gradient. In the examples presented in this paper, an influence radius of $4r$ is used to provide a smooth surface. The values computed using the method of Solenthaler et al. correspond to a signed implicit function, but do not form a signed distance field. Thus, they are converted by first triangulating the isosurface, and then setting the interface cells' value $\phi(\mathbf{x}_i)$ using their distance to the nearest triangle.

5.2. Projection



(a) First step (b) Second step

Figure 7: Our 2-step projection: (a) First, the projection $\mathbf{x}_i^*$ on the isosurface $\phi = 0$ is computed; (b) then, the final projected position $\mathbf{x}_i^{proj}$ is found by moving the vertex along the projection line to match the distance δ_i.

As stated previously, the implicit function, which is a signed distance field, and its gradient are used to project the explicit mesh vertices, in contrast to the method of Yu et al. [YWTY12], which used their normals. Furthermore, an initial distance δ_i between a vertex and the isosurface $\phi = 0$ should be maintained during the projection in order to preserve surface details. This distance is computed at time $t = 0$ for all initial mesh vertices, and at creation time for vertices created later during the animation. By definition, in a signed distance field, the opposite of the gradient $-\nabla\phi(\mathbf{x}_i)$ points in the direction of the closest point on the isosurface $\phi = 0$. Also, the value $\phi(\mathbf{x}_i)$ is equal to the distance from $\mathbf{x}_i$ to that point. Thus, the projection $\mathbf{x}_i^{\text{proj}}$ of $\mathbf{x}_i$ on the isosurface $\phi = \delta_i$ could be obtained using the following equation:

$$\mathbf{x}_i^{proj} = \mathbf{x}_i - \frac{\nabla\phi(\mathbf{x}_i)}{\|\nabla\phi(\mathbf{x}_i)\|} \cdot (\phi(\mathbf{x}_i) - \delta_i)\,. \qquad (1)$$

However, our experiments showed that Eq. 1 often fails to find a good solution when $\mathbf{x}_i$ lies inside a cell containing a local maximum or minimum of ϕ, because the grid interpolation approximation systematically under- or over-estimates the value of ϕ in these cells. To improve the precision in such cases, the proposed approach uses a two-step projection, where the closest point on $\phi = 0$ is first found (see Fig. 7(a)), and then the vertex is moved along the projection line to match the distance δ_i (see Fig. 7(b)). As the value of ϕ can be unreliable, the projection on $\phi = 0$ is found by iteratively moving toward $\phi = 0$:

$$\mathbf{x}_i^* = \mathbf{x}_i^* - \varepsilon \cdot \frac{\nabla\phi(\mathbf{x}_i)}{\|\nabla\phi(\mathbf{x}_i)\|} \cdot \phi(\mathbf{x}_i),$$

where $\mathbf{x}_i^*$ is initially set to $\mathbf{x}_i$. The coefficient ε represents the fraction of the distance to travel toward $\phi = 0$, so it should have a value in the range (0,1], and is set to 0.35 in the examples shown in this paper. This operation is repeated until $\phi(\mathbf{x}_i^*)$ is lower than a predefined threshold ($0.005\Delta x_\phi$ is used in the examples shown in this paper, where Δx_ϕ is the cell size of ϕ). During this stage, a cubic interpolation is used to compute the value of $\phi(\mathbf{x}_i)$ in order to provide a better approximation of the surface curvature during the projection. Afterward, the final projection position $\mathbf{x}_i^{\text{proj}}$ is computed using the following equation:

$$\mathbf{x}_i^{proj} = \mathbf{x}_i^* + \frac{\mathbf{x}_i - \mathbf{x}_i^*}{\|\mathbf{x}_i - \mathbf{x}_i^*\|} \cdot \delta_i.$$

After these two steps, the vertex should be correctly projected on the surface, while preserving its initial distance.

5.3. Topological changes detection for the implicit function

In order to correctly remesh the explicit surface (Sec. 7), two main types of topological changes need to be accounted for: those in the implicit surface and those in the explicit surface. Topological changes of the implicit surface are handled in this section, while those of the explicit surface will be dealt with in Sec. 7. The changes in the topology of the isosurface $\phi = 0$ that we want to detect here are those that cause the surface to merge or split. Such changes result in a sudden "jump" in the vertex position when the projection is applied, allowing the detection of these topology changes with the following threshold:

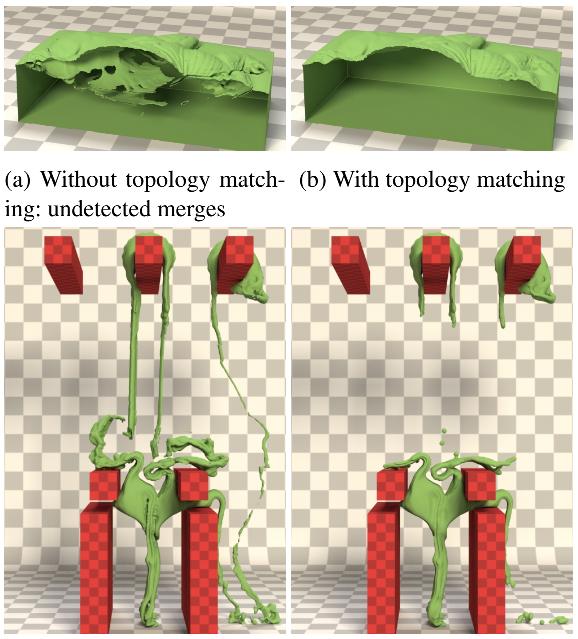
$$\left|\mathbf{x}_i - \mathbf{x}_i^{proj}\right| > \tau.$$

When a vertex exceeding this threshold is found, it is moved back to its original position $\mathbf{x}_i$, and the cells (of the Γ grid) containing the neighbor triangles of that vertex are flagged for the topology matching (see Sec. 6) and the remeshing (see Sec. 7) operations. In the examples shown in this paper, $\tau = 0.5r$.

This approach is different from the method used in the work of Yu et al. [YWTY12]. In their case, vertices were detected when the projection failed to find the isosurface $\phi = 0$. Our projection is guaranteed to find the nearest point on the isosurface $\phi = 0$, and as a result, it is necessary to use the threshold-based detection outlined above.

6. Optimization-Based Topology Matching

The goal of the topology matching stage is to handle problems outlined in Fig. 8, where the surface should merge (see Fig. 8(a)) or should disconnect (see Fig. 8(c)). These prob-


(a) Without topology matching: undetected merges (b) With topology matching

(c) Without topology matching: undetected splitting (d) With topology matching

Figure 8: Topology matching is an important step of our approach, as can be seen in the results without (left) and with (right) this stage.

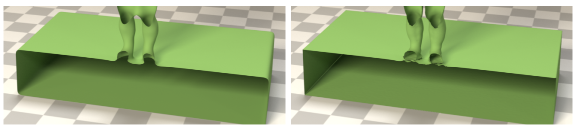

(a) Implicit surface (b) Explicit surface

Figure 9: Cutaway view showcasing the divergence between (a) the implicit and (b) the explicit surface topologies, when topology matching is not performed.

lems are caused by the divergence between the explicit and the implicit surface topologies, as can be seen in Fig. 9. Thus, it is crucial to intelligently adjust the explicit mesh to reflect the topological changes from the implicit function, while preserving its details. This will be done with an optimization that locally adjusts the topology, and globally preserves details. The topology matching stage consists of two steps. First, using the technique described by Wojtan et al. [WTGT10], a signed distance field is built from the explicit mesh, and stored in the mesh voxelization grid Γ. Then, an optimization is performed on the grid to locally adjust the topology of Γ to mirror the topological changes of ϕ, which have previously been identified during the projection (see Sec. 5.3). Note that the optimization is performed only if one or more cells have been flagged during projection. This modified mesh voxelization grid will later be used to rebuild the mesh locally (see Sec. 7).

During this operation, cells that are near the interface and have not been flagged during projection are marked as *fixed cells*. This flag indicates that these cells should not be modified by the topology matching stage. To be considered near the interface, the cell's distance from the surface should not be greater than the diagonal length of a grid cell:

$$|\Gamma_i| \le h\sqrt{3},$$

where h is the grid cell size. Afterward, all the other cells are modified to match the topology of the implicit surface, and to provide a smooth transition with the *fixed cells*.

The main idea behind the topology matching operation is that, in *non-fixed cells*, the interface $\Gamma = 0$ should lie as much as possible on the same isosurface of ϕ. Thus, the difference between the value of Γ and ϕ in those cells should be similar to that of their neighbors. This will ensure that the values for those cells of Γ follow the implicit surface topology while being consistent with the *fixed cells*. Let us define the scalar field ψ as the difference between the explicit surface voxelization Γ and the implicit function ϕ:

$$\psi(\mathbf{x}) = \Gamma(\mathbf{x}) - \phi(\mathbf{x}). \tag{2}$$

The objective is to minimize the variation of $\psi(\mathbf{x})$ around the neighborhood of $\mathbf{x}$. This can be expressed as the minimization of the Dirichlet energy function:

$$\min_{\psi} \frac{1}{2} \int \|\nabla\psi\|^2 \mathrm{d}x,$$

which is equivalent to solving the following Poisson equation:

$$\nabla \cdot \nabla\psi = 0. \tag{3}$$

The topology matching operation works by solving Eq. 3 for every *non-fixed cell*, as will be explained in the following sections.

6.1. Topology matching solver

Solving Eq. 3 first requires discretizing the scalar field ψ in a grid with the same dimensions, position and size as Γ. Thus, each vertex of ψ corresponds to a vertex of Γ, with the same position and indices. From that, Eq. 3 is reformulated using the central difference:

$$6\psi_{i,j,k} - \begin{pmatrix} \psi_{i-1,j,k} + \psi_{i+1,j,k} + \\ \psi_{i,j-1,k} + \psi_{i,j+1,k} + \\ \psi_{i,j,k-1} + \psi_{i,j,k+1} \end{pmatrix} = 0. \tag{4}$$

This equation is satisfied when $\psi_{i,j,k}$ is equal to the average value of its neighbors, resulting in a smooth scalar field. Thus, the value of ψ will be smoothly interpolated between *fixed cells*. Putting the previous equation into matrix form gives:

$$\begin{pmatrix} 6 & \alpha_{21} & \cdots & \alpha_{1n} \\ \alpha_{21} & 6 & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & 6 \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where

$$\alpha_{ij} = \begin{cases} -1 & \text{if } j \text{ is a neighbor of } i. \\ 0 & \text{otherwise.} \end{cases}$$

This is a simple linear system in the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a sparse symmetric positive definite matrix. Solving that particular type of equation is a well-known problem, and it can be carried out efficiently using a wide range of solvers. We validated our approach with both a *modified incomplete Cholesky conjugate gradient level 0 (MICCG(0))* solver and a *successive over-relaxation (SOR)* solver. While the *MICCG(0)* solver performed better with a larger number of cells, in most of our examples the speed of the *SOR* solver was similar or better because of its lower time cost per iteration. Furthermore, this solver does not require any matrix construction, which therefore lowers its memory requirement. Thus, a *SOR* solver was used in the examples shown in this paper. This solver works by iteratively updating the value of each grid vertex ψ_i using the following equation:

$$\psi_i^{l+1} = (1-\omega)\psi_j^l + \frac{\omega}{\alpha_{ii}}\left(-\sum_{j<i}\alpha_{ij}\psi_j^{l+1} - \sum_{j>i}\alpha_{ij}\psi_j^l\right). \quad (5)$$

The relaxation factor ω is used to accelerate convergence, and a value in the range of $0 < \omega < 2$ guarantees convergence. Choosing its value carefully can increase the performance of the solver. Fortunately, Yang and Gobbert [YG09] showed that the optimal value ω_{opt} for the 3D Poisson equation with Dirichlet boundary conditions can be computed using the following equation:

$$\omega_{opt}(N) = \frac{2}{1+\sin\left(\frac{\pi}{N+1}\right)},$$

where N is the dimension of the grid, assuming a $N \times N \times N$ grid. Since our grid does not necessarily have uniform dimensions, we use the maximum dimension along one axis. This equation was used to compute the relaxation parameter value in all the examples shown in this paper.

The solver is iterated until the maximum relative difference gets below a threshold:

$$\max_i \|\psi_i^{l+1} - \psi_i^l\| < \beta.$$

The threshold value $\beta = 0.01h$ is used in all the examples shown in this paper.

Algorithm 1 shows the complete topology matching stage. During the first step of this stage, ψ_i is initialized for each cell using the initial solution $\psi_i = 0$. Moreover, the *fixed cells* of Γ are identified and the Dirichlet boundary conditions are enforced for the cells in ψ (see Sec. 6.2). Note that since the grid ϕ resolution is not necessarily the same as Γ, linear interpolation is used to compute the value of ϕ at each grid vertex. Afterward, the *successive over-relaxation* iterations are computed until convergence is reached. Finally, Γ is updated using the new values of ψ.

Algorithm 1: Topology matching

```
1  Procedure TOPOLOGY_MATCHING
2      forall the cells i in Γ do
3          if (i is not flagged) AND ($\|\Gamma_i\| \leq (h\sqrt{3})$) then
4              Set i as a fixed cell
5              $\psi_i = \Gamma_i - \text{interpolate}(\mathbf{x}_i, \phi)$
6          else
7              $\psi_i = 0$
8      while ($\max_i \|\psi_i^{l+1} - \psi_i^l\| > \beta$) do
9          forall the non-fixed cells i do
10             Compute $\psi_i^{l+1}$ using Eq. 5
11     forall the non-fixed cells i do
12         $\Gamma_i = \psi_i + \text{interpolate}(\mathbf{x}_i, \phi)$
```

Note that the *successive over-relaxation* solver does not require a temporary structure to store the previous values of ψ^k. This is because ψ_i^{k+1} is computed using the previously updated value when the neighbor has already been processed.

6.2. Boundary conditions

For grid cells lying on the boundaries, the missing neighbor values are replaced by the cell value $\psi_{i,j,k}$ in Eq. 4. It can be seen that doing so only removes the contribution of those neighbors while decrementing the coefficient of $\psi_{i,j,k}$. Thus, the condition can be enforced by modifying the way α_{ii} is computed:

$$\alpha_{ii} = N_i,$$

where N_i is the number of neighbor cells of i.

In order to preserve the original surface inside the *fixed cells*, Dirichlet boundary conditions are enforced. The values $\psi_{i,j,k}$ of those *fixed cells* are initialized using Eq. 2, and are not modified by the solver.

7. Explicit Surface Topology & Meshing

The topology of the explicit mesh voxelization (Γ) and the topology of the implicit function (ϕ) now match, and the cells of Γ requiring remeshing based on changes in the implicit surface topology are marked for reconstruction. We use the method of Wojtan et al. [WTGT09] to mark the cells requiring remeshing based on changes in the topology of the explicit surface, and to reconstruct the mesh geometry inside marked cells. To summarize their method, the topological changes of the explicit mesh surface are first detected

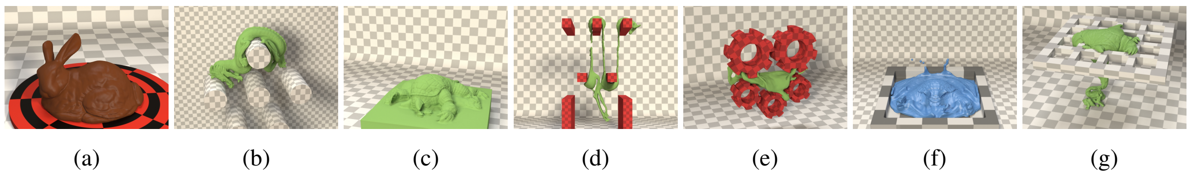


Figure 10: We tested our approach on several scenarios, ranging from slow deformation to a large number of complex topological changes.

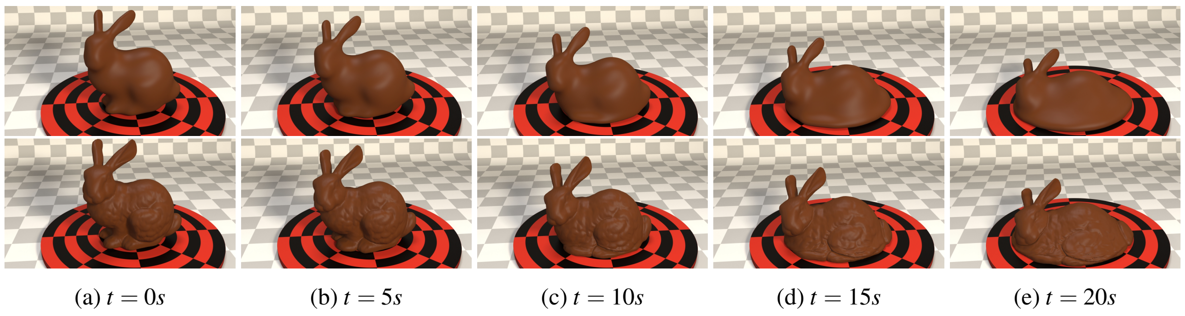


Figure 11: Melting of the Stanford bunny from the bottom using only 27,952 particles. Top: the implicit surface built from the particles; bottom: our explicit surface.

by marking cells where multiple interpenetrations occur in the explicit mesh. A "deep cell test" is then used to compare the topology of the explicit mesh and its voxelization Γ, in order to detect regions where merging and splitting occurred. Afterward, the geometry in the marked cells is removed from the explicit mesh, and then reconstructed using the marching cube algorithm from Lorensen et al. [LC87]. In our implementation, the modified look-up table of Montani et al. [MSS94] is used to resolve ambiguous cases and avoid producing cracks in the resulting geometry. Note that the detection and the surface reconstruction are based on the adjusted value of the explicit mesh voxelization Γ, ensuring a smooth transition with the surrounding detailed explicit mesh.

8. Robustness

As is typical in explicit mesh methods [BB09, WTGT09, WTGT10, YWTY12], after the mesh is advected, edges that are too small are collapsed and those that are too long are split. Also, as in the method of Yu et al. [YWTY12], edges are collapsed when the dihedral angle of the supporting triangles is too large. These operations help maintain a good mesh quality throughout the surface tracking process. As in previous methods [BB09, WTGT10], the modified butterfly subdivision scheme of Zorin et al. [ZSS96] is used to position the new mesh vertices during splitting operations. During a collapse operation, if a vertex lies on a ridge or a corner, its position is used, otherwise, the butterfly subdivision is used instead, such as in Brochu and Bridson [BB09]. We refer the reader to the course notes of Wojtan et al. [WMFB11] on mesh-based surface tracking for a more detailed explanation of how to preserve a good mesh quality.

The stitching mechanism of Wojtan et al. [WTGT10] is used when connecting the newly created mesh with the old one. This method better preserves the details in those regions than that of Wojtan et al. [WTGT09], and is less prone to the generation of a non-manifold geometry. Furthermore, as in the work of Wojtan et al. [WTGT09], when a non-manifold geometry is detected, the neighbor cells are marked and reconstructed.

9. Results

We tested our approach on a range of scenarios, ranging from a slowly melting object to severe deformations induced by rigid obstacles tearing apart objects made of viscous material. Several test scenarios can be found in the accompanying video, and representative frames are shown in Fig. 10. We first show that surface details are well preserved over time for low deformation rates by slowly melting the Stanford bunny from its base (see Fig. 11), and dropping a highly viscous dragon on obstacles (see Fig. 1, Fig. 10(b), and the video). The bunny and the dragon were simulated using only 27,952 particles and 20,456 particles, respectively, and yet our approach preserved the original details of the mesh surface throughout the simulation, while the implicit surface did not.

We then show that the approach correctly handles merging fluids by dropping viscous objects on top of one another. The cutout views in Fig. 12 and in the video show that the topo-

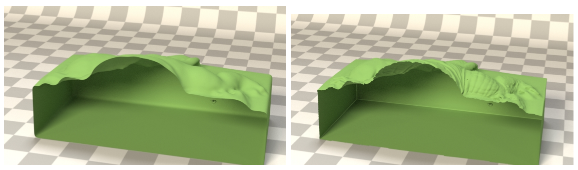


Figure 12: Viscous objects falling on top of one another at time $t = 4s$ (269,257 particles). Left: cutout view of the implicit surface built from the particles; right: cutout view of our explicit surface.

logical changes are correctly handled as the objects merge together, ensuring that the topology of the explicit mesh surface is consistent with that of the implicit surface. With our approach, the finer details of the original mesh surface, such as the horns of the gazelle and the armadillo's shell, are well preserved (see accompanying video). Similarly, we show that our approach handles fluid splitting correctly by tearing apart objects (see Fig. 13, Fig. 10(e), and the video). Surface splitting is handled gracefully by our approach, consistent with the implicit surface topology. Moreover, surface details are well preserved in regions that have not been affected by topological changes, such as the head of the gazelle.

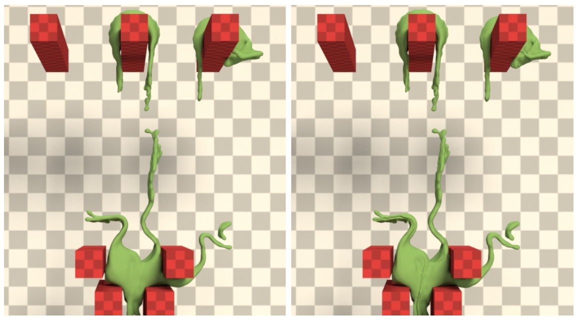
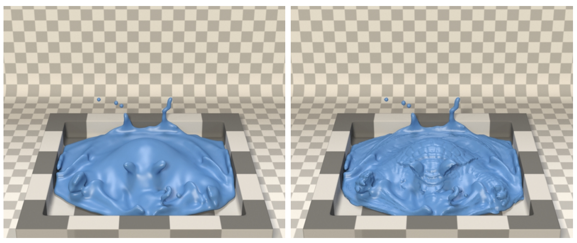


Figure 13: Tearing apart an object at time $t = 2s$ (21,362 particles). Left: the implicit surface built from the particles; right: our explicit surface.

We show that the approach handles cases where multiple and complex topological changes occur by simulating a non-viscous liquid (see Fig. 14). Surface details, such as the face, the teeth, and the shell of the armadillo, are preserved until reconstruction occurs in those areas. Furthermore, the surface is stable when multiple splittings and mergings occur at once. Even the splashes are captured correctly, thanks to our topology matching stage.

In Fig. 15, two liquid bunnies collided before falling into a pool of liquid. Despite the complexity of the topological changes, the resulting explicit mesh surface closely follows the implicit surface. Note that the complete sequence is available in the accompanying video.


(a) $t = 0.79s$

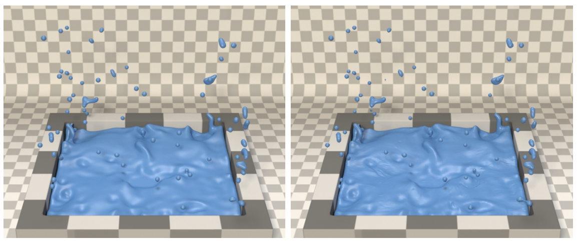

(b) $t = 2.25s$

Figure 14: Non-viscous liquid Stanford armadillo (35,214 particles). Left: the implicit surface built from the particles. Right: our explicit surface.

9.1. Comparison

We implemented the method of Yu et al. [YWTY12], and compared it to our approach for the dragon example. As can be seen in Fig. 16, their method fails to preserve surface details of the original mesh, because of the projection on an implicit surface. Our approach successfully preserves such surface details. In our experimentations, we also observed that the projection of Yu et al. [YWTY12] is unstable, and introduces flickering of the surface in regions with high curvature if it is calculated only once per frame (see accompanying video). In contrast, our projection proved to be stable using only one step per frame, which allows us to handle topological changes only once per frame.

9.2. Discussion

We compared the results using a coarser and finer resolution for the explicit mesh voxelization Γ in Fig. 17 and in the video. When the resolution is too coarse, finer details, such as the horns of the gazelle, might be erroneously split. Furthermore, the reconstruction might not be accurate enough, which can result in loss of details, and in lumps floating around the object. On the other hand, as can be seen in Table 1, the computation times of the topology matching and the explicit topology & meshing stages increase considerably when a finer resolution is used. The examples shown in this paper use $\Delta x_\Gamma = 0.5r$, except for the ones shown in Fig. 17. Smaller details could be better preserved, even with a coarser grid, by using the approach of Wojtan et al. [WTGT10] to handle the explicit surface topological

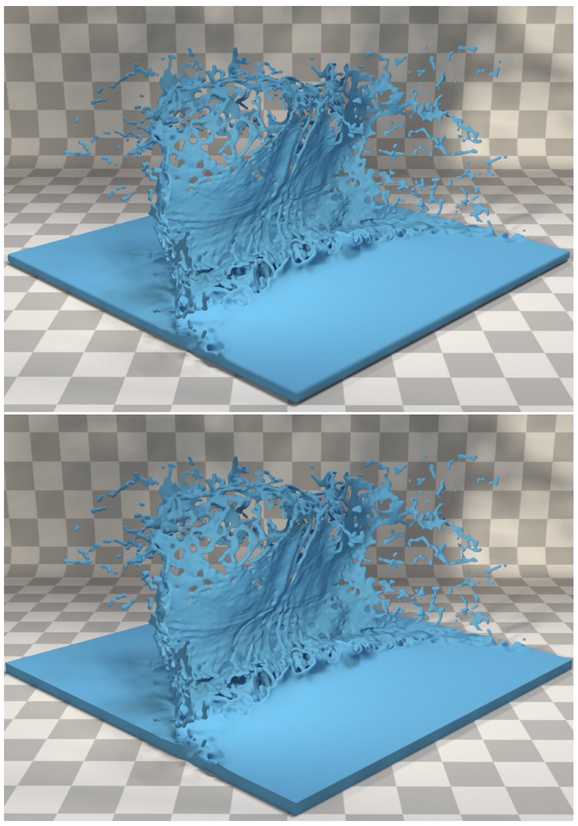


Figure 15: Splashes generated after two liquid bunnies collided. Top: the implicit surface; bottom: our explicit mesh surface.

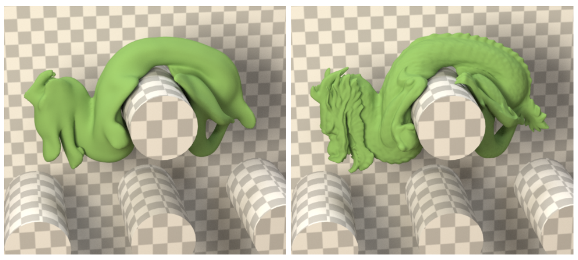


Figure 16: Comparison between the technique of Yu et al. [YWTY12] (left) and our approach (right). Our projection preserves the details of the initial explicit mesh surface.

changes. However, since that technique does not reconstruct the surface directly from the voxelization, it would also require changing the way our topology matching stage works.

Similarly, we validated our scenarios with different resolutions for the implicit function ϕ. We found that using a coarser resolution, such as $\Delta x_\phi = r$, resulted in small oscillations of the implicit surface that where transferred to the

	Topology matching	**Explicit topology & meshing**
$\Delta x_\Gamma = 0.25r$	155.79 s	36.91 s
$\Delta x_\Gamma = 0.5r$	8.28 s	5.11 s
$\Delta x_\Gamma = 1.0r$	0.53 s	1.90 s
$\Delta x_\Gamma = 2.0r$	0.05 s	1.11 s

Table 1: Timing comparison for the gazelle scenario, with different resolutions Δx_Γ.

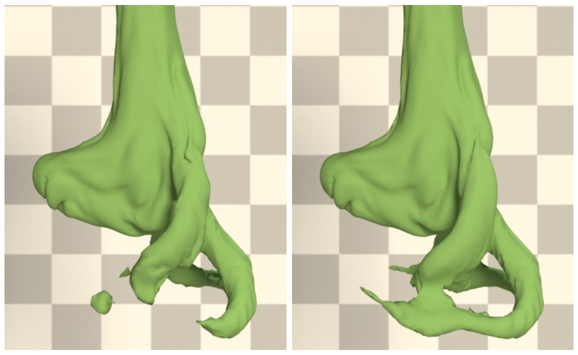


Figure 17: Comparing our results with the resolution $\Delta x_\Gamma = r$ (left) and $\Delta x_\Gamma = r/2$ (right).

explicit surface during animation. Using a finer resolution $\Delta x_\phi = r/2$ removed the oscillations.

The timings of all the examples are shown in Table 2, and the relative time for each part of the approach is depicted in Fig. 18. The most time-consuming parts of our approach are the computation of the implicit surface [SSP07], the topology matching stage, and the explicit topology & meshing, as can be seen in Fig. 18. Furthermore, the ratio of the computation times for these parts is not always the same. As can be seen in Table 2, the computation time of the implicit surface is proportional to the number of particles, but it should be noted that the number of particles does not affect the computation times of the other parts. The timings of the advection, projection, and explicit topology & meshing stages depend mostly on the number of triangles. Moreover, the computation times of the explicit topology & meshing stage are proportional to the amount of topological changes. The topology matching stage also requires more time to compute for scenarios where the surface is contained within a large volume (splitting gazelle, falling objects, and gears). As the number of cells increases, the matrix system becomes larger, and the solver requires more iterations to convergence. However, the impact this stage has on the overall time is greatly reduced as this operation is skipped when no early topological changes are detected (see Sec. 5.3). This can be observed in scenarios with few topological changes (bunny, falling objects, and viscous armadillo), where it was performed on less than 50% of the total frames. We believe computation times for this operation could be further reduced by limiting the processing to cells within a threshold distance to the *flagged*

			Avg. time per frame (s)						
Example	**Avg. # particles**	**Avg. # triangles**	**Advect**	**Implicit function**		**Proj.**	**Topology matching**	**Explicit topology & meshing**	**Total**
				[SSP07]	**SDF**				
Dragon (Fig. 1)	20,456	257,785	1.17	3.01	1.39	1.24	1.59 (58%)	3.16	11.55
Gears (Fig. 10(e))	8,877	121,414	0.39	1.82	3.06	0.48	13.48 (83%)	5.55	24.77
Viscous armadillo (Fig. 10(g))	35,195	85,274	0.41	5.34	3.43	0.32	4.13 (43%)	3.59	17.22
Melting bunny (Fig. 11)	27,952	66,862	0.30	3.98	0.92	0.24	0.00 (0%)	1.48	6.92
Falling objects (Fig. 12)	259,752	370,789	1.29	35.96	9.60	1.10	8.18 (38%)	7.27	63.40
Splitting gazelle (Fig. 13)	21,362	242,932	0.87	3.43	2.53	0.81	8.28 (83%)	5.11	21.03
Liquid armadillo (Fig. 14)	35,214	100,870	0.41	6.04	4.15	0.67	5.43 (94%)	4.51	21.20

Table 2: Timings for all examples. The per frame average computation times are for the explicit mesh advection (Sec. 4), the implicit surface evaluation and triangulation using the method of Solenthaler et al. [SSP07] (Sec. 5.1), the signed distance field calculation (voxelizing the implicit surface, and then performing the FMM [LC87]) (Sec. 5.1), the detail-preserving projection (Sec. 5.2, and Sec. 5.3), the topology matching stage (Sec. 6), and the explicit topology & meshing stage (Sec. 7). Numbers in parentheses correspond to the percentages of frames where the topology matching stage was executed.

cells. However, care must be taken not to choose a distance that is too small, which might prevent distant volumes from merging, and would result in holes on the surface.

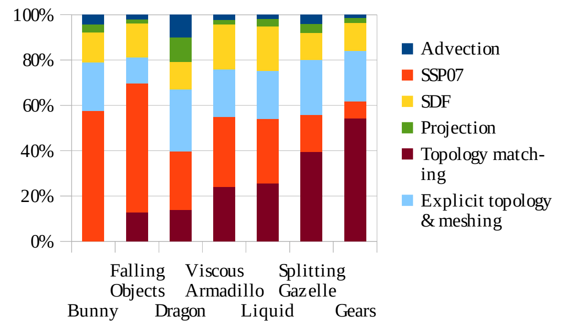


Figure 18: Timing percentages for all examples. The per frame timing percentages are for the same stages as those given in Table 2.

In our approach, all the computations are performed only once per frame. This is different from the case of Yu et al. [YWTY12], where the advection and the topological changes were performed at every timestep of the simulation. In their case, this is necessary since an accurate surface representation is needed throughout the simulation in order to compute accurate surface tension forces. Other papers based on an Eulerian simulation [WTGT09, Mül09, WTGT10] mention that they can handle topological changes only once per frame; however, they still need to advect the surface at every timestep since they need it to update the interior cells of the simulation. Even though the advection stage is not costly, this dependency makes it harder to recompute the surface without re-running the simulation. Since we use a particle-based simulation, no information from the surface is needed during the simulation. Therefore, with our approach, it is possible to generate the explicit mesh surface as a post-process operation once the entire simulation has been computed. This enables a more user-friendly workflow, in which the user can obtain the desired behavior from the simulation, and then separately generate and adjust the explicit mesh surface without affecting the simulation.

All the simulations shown in this paper were generated using Houdini's FLIP fluid solver, except for the liquid Stanford armadillo example (see Fig. 14), where an SPH simulator was used [BT07]. Particles resampling was disabled in the FLIP solver to prevent particles from being added or removed near the surface, which results in minor flickering of the implicit surface. Our approach could take advantage of parallel computing; however, the timings shown in this paper are for a sequential execution on a single core. The timings information provided in this section are for a 3.2 GHz Intel i7-3930K CPU with 32 GB of RAM. The images and videos were rendered using Houdini's Mantra.

9.3. Limitations

When large regions of the explicit mesh surface are reconstructed, temporal discontinuities may occur. These discontinuities may be in the form of a surface *popping* artifact on some frames of the animation, and may be caused by a coarse grid resolution (see Fig. 17 and the video), or by a late detection of small topological changes on the implicit surface. Nonetheless, in the first case, a smaller voxelization grid cell size reduces this problem, at the cost of longer computation times. As for the second case, the problem occurs when a small topological change on the implicit surface does not induce a vertex movement of the nearby mesh that is large enough to be detected by the threshold described in Sec. 5.3. The reconstruction is delayed until the threshold is triggered, which forces the reconstruction of a larger region of the explicit mesh. This problem generally occurs when small holes are created in a thin volume of liquid. This can be seen in the viscous armadillo (see Fig. 10(g) and the video) and in the colliding bunnies (see Fig. 15 and the video) scenarios, where holes in the implicit surface are not immediately detected and handled, resulting in a delayed reconstruction of a larger region of the mesh. This can be reduced by using a smaller threshold τ when detecting topological changes in the implicit function. However, choosing a value that is too small might unnecessarily reconstruct portions of the mesh where no topological changes of the implicit function occurred. Furthermore, changes in topology that occur further from the explicit surface, such as the creation of bubbles and droplets, may not be detected by our criterion. However, these features are transferred to the mesh voxelization Γ during the topology matching stage. Because this stage is triggered only when a topological change is detected by the criterion, in some cases it may require a few frames before the change in topology is reflected on the explicit mesh. A compromise is to perform the topology matching stage at every frame, at the cost of longer computation times. The problem of identifying topological changes of an implicit surface is not new, and more robust approaches could be used instead of our criterion. For instance, it can be shown from Morse theory that changes in the topology of a parameterized implicit surface occur at the location of a critical point [Har98]. By tracking such events using interval arithmetic, Stander and Hart [SH97] were able to identify topological changes of an implicit surface. While such an approach would add a significant overhead to our computation times, we believe it could allow the early detection of holes, bubbles, and splashes. One could also use concepts from digital topology to detect such changes. Using a level-set evolution such as that of Han et al. [HXP03], which preserves topology, our implicit surface could be morphed into the explicit mesh voxelization while preserving its topology. The conflicting regions between the two surfaces could then be used to identify where their topologies differ. It should be noted that such a technique would increase computation times, and that developing a robust solution might not be as trivial as expected. The evolution of the implicit surface needs to be carefully directed, as multiple solutions can be topologically equivalent at the global scale, but might not reflect the same local topological changes.

While our approach works well with viscous fluids, some limitations have however been observed with non-viscous fluids. In the latter cases, it might be undesirable to preserve all surface details. This can be seen in the liquid armadillo (see Fig. 14 and the video) and splashing bunnies (see Fig. 15 and the video) examples, where small wrinkles are preserved on the liquid surface as it comes to rest, and small folds are sometimes preserved where two surfaces merge. A smoother surface is expected for such a non-viscous liquid. Decreasing the distance δ_i between the vertices and the isosurface should result in the expected behavior, as the explicit surface would then take on the appearance of the isosurface. A metric based on the viscosity and the deformation undergone by the surface, e.g., the magnitude of the pressure gradient, such as in the work of Goldade et al. [GBW16], could be used to determine where and when to adjust δ_i. Additionally, while our explicit surface can handle the frequent remeshing caused by the multiple topological changes that non-viscous liquids generally undergo, it tends to lose details at points where topological changes of the implicit surface occurred. This is caused by the topology matching stage, which tries to replicate the shape of the implicit surface in regions that are being reconstructed. Nonetheless, our approach is still relevant for non-viscous liquids, as the explicit mesh surface preserves its original shape until it undergoes several topological changes, and can also be used to track surface properties such as color or physical quantities, such as in the work of Yu et al. [YWTY12].

10. Conclusion

We presented a new detail-preserving projection, coupled with a novel topology matching stage. Unlike previous methods, our approach preserves the original surface details, even when the underlying simulation is very coarse. The approach was successfully used with simulations of melting objects, highly viscous objects, and even non-viscous liquids. It was used with a FLIP and an SPH simulator, and could easily be adapted to any particle-based simulator.

It would be interesting to extend the approach to preserve thin sheets even when using a coarse grid resolution. We would also like to extend the approach to better handle non-viscous liquids in order to preserve the surface details and smooth them out as the surface changes. Furthermore, this approach could be extended to accurately track surface properties such as color or texture coordinates.

Acknowledgements

This work was supported by CINQ, FQRNT, GRAND, NSERC, Mokko Studio, and École de technologie supérieure. We would like to thank Mokko Studio for the men and the gazelle models, and the help with the renders. We would also like to thank Side Effects Software for providing us with Houdini licenses.

References

[APKG07] Adams B., Pauly M., Keiser R., Guibas L. J.: Adaptively sampled particle fluids. *ACM Trans. Graph. 26*, 3 (July 2007). 3

[ATW13] Ando R., Thürey N., Wojtan C.: Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph. 32*, 4 (July 2013), 103:1–103:10. 3

[BB09] Brochu T., Bridson R.: Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing 31*, 4 (2009), 2472–2493. 3, 9

[BGB11] Bhatacharya H., Gao Y., Bargteil A.: A level-set method for skinning animated particle data. In *Proc. of the*

2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2011), SCA '11, ACM, pp. 17–24. 3

[BT07] BECKER M., TESCHNER M.: Weakly compressible SPH for free surface flows. In *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007), SCA '07, Eurographics Association, pp. 209–217. 12

[DBG14] DA F., BATTY C., GRINSPUN E.: Multimaterial mesh-based surface tracking. *ACM Trans. Graph. 33*, 4 (July 2014), 112:1–112:11. 3

[EB14] EDWARDS E., BRIDSON R.: Detailed water with coarse grids: Combining surface meshes and adaptive discontinuous galerkin. *ACM Trans. Graph. 33*, 4 (July 2014), 136:1–136:9. 3

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

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *Proc. of SIGGRAPH '01* (2001), ACM, pp. 23–30. 2

[GBW16] GOLDADE R., BATTY C., WOJTAN C.: A practical method for high-resolution embedded liquid surfaces. *Computer Graphics Forum 35*, 2 (2016), 233–242. 13

[Har98] HART J. C.: *Morse Theory for Implicit Surface Modeling*. Springer Berlin Heidelberg, 1998, pp. 257–268. 13

[HXP03] HAN X., XU C., PRINCE J. L.: A topology preserving level set method for geometric deformable models. *IEEE Trans. on Pattern Analysis and Machine Intelligence 25*, 6 (June 2003), 755–768. 13

[KSK09] KIM D., SONG O.-Y., KO H.-S.: Stretching and wiggling liquids. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 120:1–120:7. 2

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 163–169. 2, 9, 12

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 457–462. 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), SCA '03, Eurographics Association, pp. 154–159. 3, 4

[MSS94] MONTANI C., SCATENI R., SCOPIGNO R.: A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer 10*, 6 (1994), 353–355. 9

[Mül09] MÜLLER M.: Fast and robust tracking of fluid surfaces. In *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), SCA '09, ACM, pp. 237–245. 3, 12

[Set95] SETHIAN J. A.: A fast marching level set method for monotonically advancing fronts. In *Proc. Nat. Acad. Sci* (1995), pp. 1591–1595. 5

[SH97] STANDER B. T., HART J. C.: Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proc. of SIGGRAPH 97* (1997), Annual Conference Series, pp. 279–286. 13

[SSP07] SOLENTHALER B., SCHLÄFLI J., PAJAROLA R.: A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds 18*, 1 (2007), 69–82. 2, 3, 5, 11, 12

[Wil08] WILLIAMS B.: *Fluid surface reconstruction from particles*. Master's thesis, The University of British Columbia, 2008. 3

[WMFB11] WOJTAN C., MÜLLER-FISCHER M., BROCHU T.: Liquid simulation with mesh-based surface tracking. In *ACM SIGGRAPH 2011 Courses* (2011), ACM, pp. 8:1–8:84. 2, 9

[WTGT09] WOJTAN C., THÜREY N., GROSS M., TURK G.: Deforming meshes that split and merge. *ACM Trans. Graph. 28*, 3 (July 2009), 76:1–76:10. 3, 4, 8, 9, 12

[WTGT10] WOJTAN C., THÜREY N., GROSS M., TURK G.: Physics-inspired topology changes for thin fluid features. *ACM Trans. Graph. 29*, 4 (July 2010), 50:1–50:8. 3, 7, 9, 10, 12

[YG09] YANG S., GOBBERT M. K.: The optimal relaxation parameter for the SOR method applied to the Poisson equation in any space dimensions. *Applied Mathematics Letters 22*, 3 (Mar. 2009), 325–331. 8

[YT13] YU J., TURK G.: Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Trans. Graph. 32*, 1 (Feb. 2013), 5:1–5:12. 3

[YWTY12] YU J., WOJTAN C., TURK G., YAP C.: Explicit mesh surfaces for particle based fluids. *Comp. Graph. Forum 31* (May 2012), 815–824. 2, 3, 4, 5, 6, 9, 10, 11, 12, 13

[ZB05] ZHU Y., BRIDSON R.: Animating sand as a fluid. *ACM Trans. Graph. 24*, 3 (July 2005), 965–972. 1, 3

[ZSS96] ZORIN D., SCHRÖDER P., SWELDENS W.: Interpolating subdivision for meshes with arbitrary topology. In *Proc. of SIGGRAPH '96* (1996), ACM, pp. 189–192. 9