Vision, Modeling, and Visualization (2023)
T. Grosch and M. Guthe (Eds.)

Weighted Laplacian Smoothing for Surface Reconstruction of Particle-based Fluids

Fabian Löschner[†], Timna Böttcher[†], Stefan Rhys Jeske and Jan Bender

RWTH Aachen University, Germany

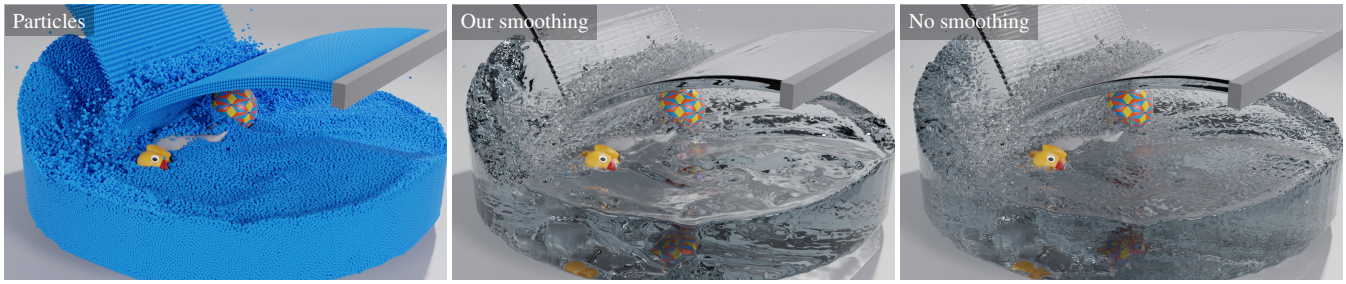


Figure 1: *We demonstrate the effectiveness of our post-processing approach in a scene with dynamic rigid bodies (1 million fluid particles). The images show the particles (left), surface reconstruction with our proposed smoothing (middle) and surface reconstruction without smoothing (right). Our method significantly reduces the bumpiness of the surface while preserving splashes and isolated particles.*

Abstract

In physically-based animation, producing detailed and realistic surface reconstructions for rendering is an important part of a simulation pipeline for particle-based fluids. In this paper we propose a post-processing approach to obtain smooth surfaces from "blobby" marching cubes triangulations without visual volume loss or shrinkage of drops and splashes. In contrast to other state-of-the-art methods that often require changes to the entire reconstruction pipeline our approach is easy to implement and less computationally expensive.

The main component is Laplacian mesh smoothing with our proposed feature weights that dampen the smoothing in regions of the mesh with splashes and isolated particles without reducing effectiveness in regions that are supposed to be flat. In addition, we suggest a specialized decimation procedure to avoid artifacts due to low-quality triangle configurations generated by marching cubes and a normal smoothing pass to further increase quality when visualizing the mesh with physically-based rendering. For improved computational efficiency of the method, we outline the option of integrating computation of our weights into an existing reconstruction pipeline as most involved quantities are already known during reconstruction. Finally, we evaluate our post-processing implementation on high-resolution smoothed particle hydrodynamics (SPH) simulations.

CCS Concepts

• ***Computing methodologies*** → ***Shape modeling;*** *Animation;*

1. Introduction

Particle-based methods are often used to simulate fluids, especially in the field of physically-based animation. These methods are able to model flow behavior without having to explicitly track surfaces and can therefore efficiently simulate phenomena with complex fluid boundaries, like splashes or interaction with solids of arbitrary shape. One of the most popular particle-based methods is Smoothed Particle Hydrodynamics (SPH) [GM77, KBST22], which can be used for a wide range of materials, such as fluids [BK17], elastic solids [KBFF*21] or snow [GHB*20]. However, visualizing the particle-based fluid as a continuous surface is not a straightforward process, as most rendering methods need an explicit representation of the fluid surface.

† equal contribution

DOI: 10.2312/vmv.20231245

Extracting high quality surfaces from the simulated particle data has proven to be a challenging task. Marching cubes (MC) produces meshes with bad topology and irregularly placed particles cause bumps in the reconstructed surface. State-of-the-art methods use anisotropic kernels [YT13] or constrained optimization [BGB15] to solve this problem, but are harder to implement, involve a lot of parameters and are computationally expensive.

Instead of modifying the surface reconstruction it is also possible to remove the bumps in a post-processing step. Applying Laplacian smoothing does result in smooth surfaces, but at the cost of severe volume shrinkage, causing smaller drops to collapse to points. While volume preserving methods like Taubin smoothing [Tau95] or HC smoothing [VMM99] exist, those act as low-pass filters removing high-frequency noise and are therefore not able to efficiently flatten out the planar regions while preserving detail of similar scale in complex parts of the mesh. Akinci et al. [AAIT12] propose a post-processing pipeline designed specifically for particle-based fluids, based on adaptive decimation followed by subdivision. This method is again computationally expensive.

We propose a new post-processing pipeline for MC surfaces using weighted Laplacian smoothing, retaining the simplicity of the method while solving the main issues with minimal effort. The new method is based on the observation, that features we want to preserve occur in regions with fewer particles. Intuitively, we use a quantity based on the number of particle neighbors to drive the amount of smoothing for each vertex. We further noticed how specific triangle configurations created by standard MC can lead to artifacts during smoothing. As these configurations follow certain patterns, we show how they can be detected and removed using few well-defined edge collapses. Lastly, we apply an additional post-processing step, in which smoothed normals are computed for each vertex, causing the remaining bumps to become less noticeable when using physically-based rendering. We evaluate our pipeline on SPH fluid simulations, while using standard MC to construct the surfaces, an example of which is shown in Fig. 1.

2. Related Work

The problem of producing realistic surfaces from particle data is well known in the field of visual computing and many approaches have been developed to solve the previously stated issues. While some change the definition of the surface, others focus on the mesh generation step by improving the MC algorithm. It is also possible to enhance the quality of the reconstructed surface using mesh-based post-processing approaches.

Depending on the application, different surface descriptions of particle-based fluids can be beneficial. Surface tracking methods which, e.g., evolve a mesh during simulation [YWTY12] are mostly motivated by the requirements of specific physical fluid models (e.g., surface tension). For post-processing, implicit surface-based approaches are typically more efficient. Alternatively, surface reconstruction restricted to screen-space [MSD07] can be very efficient especially for real-time applications, but requires tight coupling with the rendering pipeline for practical workflows. Instead, we focus on offline reconstruction of the entire fluid surface from an implicit surface definition.

Surface definition In computer graphics, the rendering of deforming volumes using an implicit surface was pioneered by the "metaballs" approach introduced by Blinn [Bli82]. In this spirit, most methods for particle-based surface reconstruction define the fluid-surface as the isosurface of a scalar field in 3D space. One of the most straightforward definitions is the "color field", where each particle is assigned the value of 1, which is then interpolated over the domain [MCG03]. As a result, the surface closely follows the particles but due to their varying distances, bumps will appear.

Zhu and Bridson [ZB05] instead define the surface as the zero level-set of a signed distance field, based locally on weighted averages of the particle positions. Building on this, Adams et al. [APKG07] propose to propagate the particles' distances to the surface between timesteps and to use them in the computation of the distance field.

Another way to alleviate bumpiness is to define the surface as a level-set satisfying a constrained optimization problem, which minimizes the thin-plate energy. In these approaches, the surface is bound to lie in between two signed distance fields, one corresponding to the minimum and the other to the maximum desired distance to the particles [Wil08, BGB15].

Yu and Turk [YT13] propose to use anisotropic kernels when defining the scalar field, to not only reproduce smooth flat surfaces but also extract sharp features from the data set. This is achieved by scaling the kernel functions based on a principal component analysis performed for the covariance matrix of each particle.

The main drawbacks of these more sophisticated approaches include increased computational cost, the need to tune multiple parameters and much higher implementation complexity.

Marching Cubes The most popular method to construct a triangle mesh for the isosurface of a scalar field is the Marching Cubes (MC) algorithm [LC87]. It partitions the domain using a regular cube grid and the scalar field values are computed only at its vertices. The popularity of standard MC can in part be attributed to its relative simplicity, as the grid consists of uniform cubes, which allows for intuitive indexing of vertices based on their positions.

Efficiency can be improved by computing the scalar field values only at grid points which lie close to surface particles [AIAT12]. Additionally, the triangle quality of the produced meshes can be improved. Raman and Wenger [RW08] propose the use of an extended lookup table, to avoid generating short edges due to the surface intersecting a grid cube close to a corner, however this can result in non-manifold meshes. An alternative approach with similar goals based on displacement and snapping was proposed by Moore and Warren [MW91, MW92].

Dual methods place mesh vertices inside the grid cubes instead of on the grid edges [Gib98, JLSW02]. While the resulting meshes usually have better topology and represent sharp features better, finding optimal positions for the mesh vertices is more complicated and benefits for fluids are not clear. Another family of methods is marching tetrahedra [DK91, GH95] which evolved from works aiming to solve ambiguities in the original marching cubes formulation [Blo88]. Williams [Wil08] proposes to partition the domain of particle-based fluids with specific tilings of tetrahedra which

eliminates mesh vertices with a valence of four. These vertices were identified to cause issues when Laplacian operators are applied to triangle meshes. However, this leads to less intuitive indexing of grid vertices and requires more implementation effort.

For a more general overview of surface reconstruction methods, we refer to the survey of de Araújo et al. [dALJ*15].

Post-processing To remove bumps and noise from a surface mesh, a large variety of smoothing methods can be found in the literature. Laplacian smoothing and its variants are arguably the most commonly used family of methods due to their simplicity and effectiveness.

Depending on the application of Laplacian smoothing, different combinations of temporal and spatial discretizations of the underlying diffusion equation can be chosen [BKP*10]. For example, the most simple variant is uniform Laplacian smoothing with explicit timestepping which iteratively moves each vertex to the mean position of its neighbors. While it is cheap to evaluate, the uniform Laplacian is a relatively inaccurate discretization on irregular meshes resulting in tangential drift. Alternatively, the cotangent Laplacian assigns higher weights to shorter edges and is often used to avoid this drift (see, e.g., [DMSB99]). In our application however, tangential drift could also be seen as a welcome side effect for MC meshes, as it leads to more uniform triangles.

The main drawback of Laplacian smoothing on its own is loss of detailed features and volume shrinkage. For fluid surface meshes this can cause droplets to collapse and degenerate after only a few iterations. Volume preserving approaches [Tau95, VMM99] have shown to be very successful for meshes resulting from laser scans. These methods act as a low-pass filter and remove high-frequency noise, while the overall shape of the object remains almost unchanged. Other methods assign smoothing weights based on vertex curvature [NISA06], causing vertices representing sharp features to be smoothed less. Unfortunately, none of these methods work particularly well for particle-based fluids, as the bumps in the surface have a similar scale as the original particles. Therefore, they cannot be considered high-frequency noise, nor do they necessarily have much more local curvature than isolated droplets.

Instead of smoothing, Akinci [Aki14] proposes to use incremental decimation followed by subdivision to improve the surface quality. This method is less efficient, since incremental decimation is much more expensive than explicit smoothing and cannot be parallelized easily.

Our method uses Laplacian smoothing with positional weights, similar to the vertex curvature-based approach by Nealen et al. [NISA06]. We do not want to lose details in high curvature regions of drops and splashes, and therefore apply smoothing only in areas with a lot of neighboring particles. Additionally, we use a normal smoothing approach similar to the method of Taubin [Tau01] to make the bumps less noticeable when rendering the mesh, but without further updating the vertex positions.

3. Method

Our approach differs from other smoothing methods by using neighborhood information from the original particle set to drive the amount of smoothing for each vertex. The main steps of the overall surface reconstruction pipeline are mesh generation using MC, computation of the positional weights and the smoothing itself. We also show how certain triangle configurations leading to artifacts during smoothing can be removed using specialized decimation. As a last step, we detail how we compute a smooth normal field.

3.1. Preliminaries

Before introducing our post-processing steps, we first summarize how an initial fluid surface can be reconstructed followed by the basics relevant for our smoothing approach.

Marching Cubes Reconstruction Our post-processing pipeline is designed for meshes generated using MC. As we want to show the full capabilities of our pipeline, we assume a standard implementation of the reconstruction procedure, without special modifications to remove bumps from the surface or improve triangle quality.

For this purpose, we use the color field based definition of the fluid surface combined with SPH interpolation to compute the level-set function value Φ at the MC grid points. Given a particle set $\mathcal{P}$, a grid point $\mathbf{x}_i$, an isosurface threshold t and a kernel W with smoothing length h we get:

$$\Phi(\mathbf{x}_i) = \sum_{j \in \mathcal{P}} \frac{m_j}{\rho_j} W(\mathbf{x}_i - \mathbf{x}_j, h) - t\,, \tag{1}$$

where m_j is the mass of particle j and ρ_j is its density, which is determined by SPH interpolation as well. Since the kernel is zero for all particles farther away than a predefined cutoff distance, this sum only needs to consider particles close to the grid point.

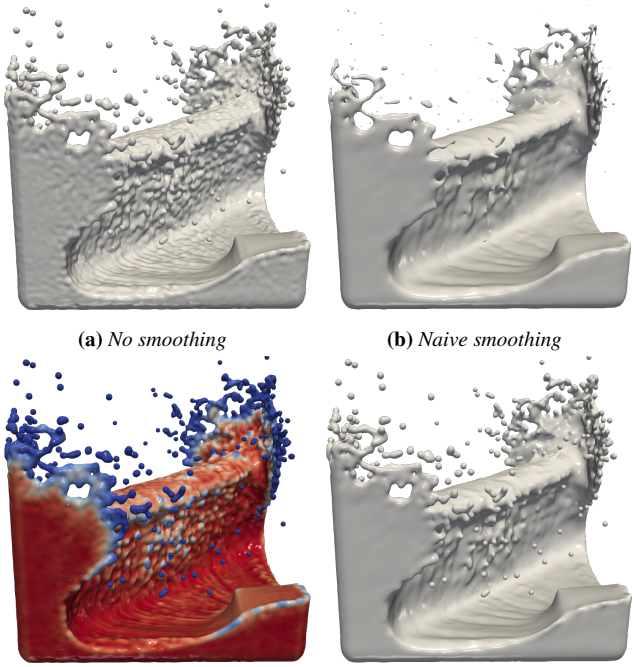
Given the level-set values at all grid points, we use MC [LC87] to obtain a triangulation. Here, the vertices of each cube are marked as either inside or outside the surface according to their sign, resulting in one of $2^8 = 256$ configurations. On each edge with a sign change, an isosurface vertex is placed, and a lookup table is used to determine the corresponding triangulation. The surface vertex positions themselves can be approximated using linear interpolation:

$$\mathbf{x} = \mathbf{x}_a + \frac{t - \phi(\mathbf{x}_a)}{\phi(\mathbf{x}_b) - \phi(\mathbf{x}_a)} (\mathbf{x}_b - \mathbf{x}_a)\,, \tag{2}$$

where $\mathbf{x}$ is the estimated position of the surface vertex, $\mathbf{x}_a$ is the grid point above and $\mathbf{x}_b$ is the one below the surface.

As a post-processing method, our approach does not depend on modifications or direct coupling to the MC implementation, and we will therefore consider it to be a black box in the following sections. As inputs we only expect a closed mesh without non-manifold edges or vertices representing the fluid surface, along with the original particle set.

Explicit Laplacian Smoothing Laplacian smoothing methods aim to improve mesh quality by moving vertices based on the positions of their neighbors. Given a mesh $M = \{E, V\}$, with edges E and vertices V, the neighborhood $\mathcal{V}_i$ of vertex $i \in V$ is defined as the set of all vertices it shares an edge with. In explicit Laplacian smoothing, each vertex is iteratively moved towards a weighted average position of its neighbors. The corresponding update vector



(a) *No smoothing* **(b)** *Naive smoothing*

(c) *Our smoothing weights* **(d)** *Our smoothing with weights*

Figure 2: *Application of 25 uniform Laplacian smoothing iterations leads to loss of volume especially for splashes and isolated particles (b). Our neighbor count based weights (c) are evaluated on the mesh ranging from 0 (blue) to 1 (red). 25 smoothing iterations with the weights result in smooth surfaces for flat regions while splashes are preserved (d).*

for a vertex i with position $\mathbf{x}_i$ is computed as:

$$\Delta\mathbf{x}_i = \sum_{j\in\mathcal{V}_i} w_{ij}(\mathbf{x}_j - \mathbf{x}_i) \quad \text{with} \quad \sum_j w_{ij} = 1\,. \tag{3}$$

The weights w_{ij} are associated with the edge (i,j). For the uniform Laplacian, the weights are given by $w_{ij} = 1/|\mathcal{V}_i|$ and depend only on the valence $|\mathcal{V}_i|$ of the vertex being updated. The new position of the vertex for iteration $n+1$ is then set to:

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \beta\Delta\mathbf{x}_i^n\,, \tag{4}$$

where β is a scaling factor between 0 and 1. A small β will decrease convergence speed, as the vertices are only moved by short distances corresponding to a smaller timestep or smaller diffusion coefficient. For $\beta = 1$ the vertex would move directly to the mean position of its neighbors, but as all vertices are updated in each iteration, this is not always optimal. Especially in noisy meshes, the neighbors might move in the opposite direction as the vertex itself causing overshooting.

With these preliminaries in mind, we can now introduce our proposed post-processing pipeline.

3.2. Our Particle Neighborhood-Based Smoothing

Applying uniform Laplacian smoothing directly to the MC surface reconstruction of a particle-based fluid can lead to severe shrinkage of the fluid volume and loss of isolated particles. The uniform Laplacian has the biggest effect in strongly convex vertex neighborhoods. Therefore, with an increasing number of smoothing iterations single particles rapidly collapse into points followed by volume loss in thin "splash" regions (see Fig. 2b).

To avoid these artifacts, we suggest employing positional feature weights $\lambda_i \in [0,1]$ (see e.g. [NISA06]) that dampen the influence of the Laplacian smoothing in the affected regions:

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \lambda_i\Delta\mathbf{x}_i^n \quad \text{with} \quad \lambda_i \in [0,1]\,. \tag{5}$$

We observe that artifacts due to loss of volume start to appear most quickly in regions with high particle neighborhood deficiencies such as isolated particles, splashes and "corners" or "edges" of the fluid between free surfaces of the fluid and boundaries. Ideally, vertices originating from the reconstruction of single particles should only be smoothed very little or not at all, whereas vertices on the surface of a half-space filled with particles should experience the maximum amount of Laplacian smoothing in order to quickly approach a plane without any bumps. This suggests using feature weights that are proportional to some measure of neighborhood deficiency in the fluid.

We suggest defining weights proportional to the number of particles in a neighborhood as this is a quantity that is intuitive and easy to compute. Per particle this quantity would be given trivially by

$$N_j = \sum_{i\in\mathcal{N}_j} 1\,, \tag{6}$$

where $\mathcal{N}_j$ is the set of neighbor particles in a radius r around particle j excluding j itself. However, the weights must be defined at the vertices of the reconstructed mesh. We chose to use standard SPH interpolation with 0^{th}-order correction to define a field that can be evaluated at the vertex positions given by

$$N(\mathbf{x}) = \frac{\sum_{j\in\mathcal{N}(\mathbf{x})} \frac{m_j}{\rho_j} N_j W(\mathbf{x}-\mathbf{x}_j,h)}{\sum_{j\in\mathcal{N}(\mathbf{x})} \frac{m_j}{\rho_j} W(\mathbf{x}-\mathbf{x}_j,h)}\,. \tag{7}$$

To obtain actual weights from this, the field $N(\mathbf{x})$ has to be mapped to the interval $[0,1]$. Using the number of expected neighbors in a half-space configuration N_{HS} (where we want to apply the most smoothing), we normalize and clamp the field using

$$N_1(\mathbf{x}) = \max\left\{\frac{N(\mathbf{x})}{N_{\text{HS}}}, 1\right\}\,. \tag{8}$$

For nearly incompressible fluids and constant particle size simulations, the value N_{HS} can be determined once for a specific choice of SPH kernel (including ratio of particle radius to smoothing length) and isosurface threshold and then re-used between simulations. In our experience these are parameters that are seldomly changed when the user settled for a specific fluid simulation framework. For compressible simulations and simulations with varying particle sizes, an automatic procedure or formula to determine this value would be preferable. Optionally, we can apply a smoothstep function such as

$$S(x) = 6x^5 - 15x^4 + 10x^3 \tag{9}$$

on top, in order push weights closer to the extremes of the range (i.e., to smooth vertices with almost no neighbors even less and

vertices on an almost flat surface even more). Finally, the weight of a mesh vertex i is then given by

$$\lambda_i = S(N_1(\mathbf{x}_i)).$$

Improving temporal coherence Directly using the number of neighboring particles as given by Eq. (6) can lead to jumps over time in the weights when particles leave each other's neighborhoods. This could potentially lead to visible "popping" artifacts between frames if the same region is smoothed with temporally discontinuous weights. Therefore, we suggest to weight neighbors inversely proportional to their distance from a given particle, similar to the effect of an SPH kernel. However, we do not require any differentiability or normalization properties for this purpose, so we suggest using a simple hat-function instead of a smooth kernel:

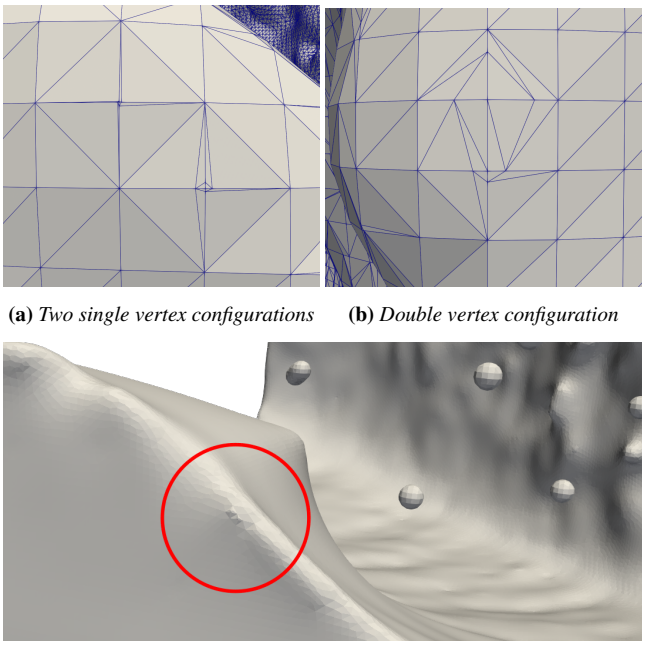
$$H(r) = \begin{cases} 1 - \frac{|r|}{h} & \text{if} \quad |r| < h, \\ 0 & \text{else} \end{cases} \tag{10}$$

with the resulting per-particle weighted neighbor count given by

$$\tilde{N}_j = \sum_{i \in \mathcal{N}_j} H(\|\mathbf{x}_i - \mathbf{x}_j\|). \tag{11}$$

To correctly normalize and clamp the resulting field, an updated half-space threshold $\tilde{N}_{\text{HS}}$ has to be determined, respectively.

These weights vary relatively smooth over a typical mesh surface and are trivially zero for isolated particles. Fig. 2 visualizes the effect of this core component of our smoothing procedure.



(a) *Two single vertex configurations* **(b)** *Double vertex configuration*

(c) *"Barnacle" artifacts occur during smoothing*

Figure 3: *Examples for "bad" MC triangulations (a) and (b) generated when the isosurface threshold is crossed close to a MC grid point (not shown) that can cause artifacts during smoothing (c).*

3.3. Specialized Decimation

While the results from our weighted smoothing already look quite promising, we occasionally encounter artifacts due to highly varying triangle quality in the MC output mesh. In particular, when the isosurface threshold is crossed very close to a vertex of the MC grid, the resulting triangulation of the neighborhood typically contains triangles that are very stretched and very small in comparison to the MC edge length. Examples for such low-quality triangle configurations are shown in Fig. 3. As the displacement computed in Laplacian smoothing depends on the edge lengths between neighboring vertices, this leads to vertices near such configurations experiencing much less smoothing over multiple iterations in comparison to vertices further away. This manifests in artifacts that we dubbed "barnacles" due to their shape in comparison to the surrounding area as shown in Fig. 3.

With uniform Laplacian smoothing it is hard to avoid this problem. While the cotangent Laplacian eliminates the strong tangential drift in these configurations, it still suffers from reduced movement of the affected vertices in normal direction in comparison to more regularly triangulated regions of the mesh. Without more elaborate and possibly more expensive timestepping, these artifacts still occur for the cotangent Laplacian and appear even sharper due to the missing relaxation of the triangulation. The core of the problem is the missing information at the center vertex about a sufficiently large mesh region (relative to surface details) due to the small incident edge lengths. Bi-Laplacian smoothing (see e.g. [KCVS98]) can give slightly better results here as it considers a two-ring of neighbors and therefore takes vertices into account that lie outside of the problematic triangle configuration. Nevertheless, it is not sufficient to get rid of the artifacts completely.

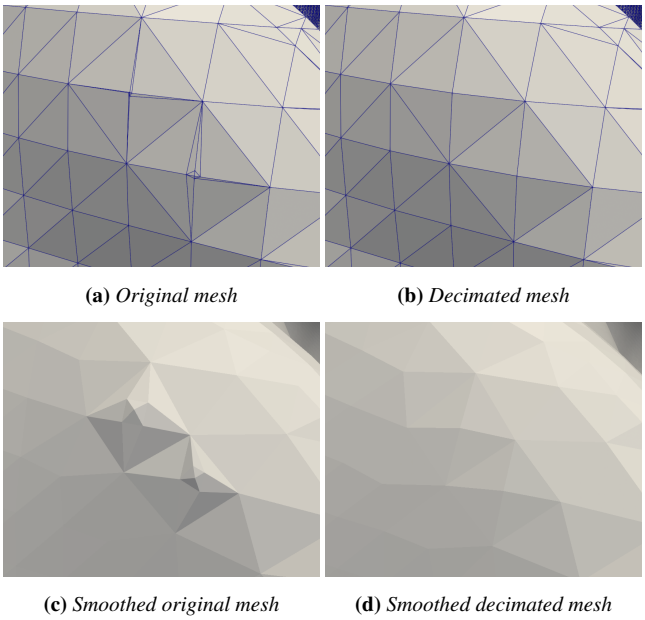
Therefore, we pursue a different approach and want to remove problematic triangulations from the MC output mesh. Let us first consider specifically decimating only problematic triangle configurations, as this is potentially more efficient than a general-purpose incremental decimation approach (see e.g. [BKP*10]).

By visual inspection of reconstructed surfaces, we noticed that the strongest "barnacle" artifacts are mostly caused by two specific triangle configurations. The configurations are similar to the "single vertex" and "double vertex" configurations shown in Fig. 3. Larger "plateaus" surrounded by stretched triangles also occur but usually do not produce artifacts as strong as the two configurations highlighted here. The configurations can be identified by their vertex connectivity patterns which can be described as follows:

- Single vertex configuration: one center vertex i with valence of $|\mathcal{V}_i| = 4$ with each connected vertex j with valence $|\mathcal{V}_j| \in \{4,5,6\}$ and $\sum |\mathcal{V}_j| = 20$.
- Double vertex configuration: two directly connected center vertices i and j with valence of $|\mathcal{V}_i| = |\mathcal{V}_j| = 5$, two shared neighbor vertices with valence of 6 and per center vertex two additional neighbor vertices with valence of 5 respectively (see Fig. 5).

We decided on this connectivity-based detection because it leads to significantly less candidates that do not create artifacts than if we would instead filter, e.g., solely based on small edge lengths.

The detection of these configurations can be performed in parallel. To decimate the problematic configurations, we then perform



(a) *Original mesh* **(b)** *Decimated mesh*

(c) *Smoothed original mesh* **(d)** *Smoothed decimated mesh*

Figure 4: *Applying our decimation procedure prevents "barnacle" artifacts from occurring during smoothing.*

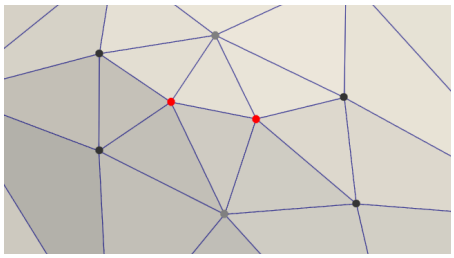


Figure 5: *Prototype "double vertex" triangle configuration with the two center vertices i and j (target vertices for the collapse operation) marked in red.*

sequential half-edge collapses of all vertices incident to the center vertices of these configurations towards the respective center vertex itself. As we only perform legal half-edge collapses, meshes after the decimation do not contain any holes, non-manifold edges or vertices. The total number of detected configurations is typically quite low (usually around 0.1% of the number of vertices in the mesh) but the procedure reliably prevents the aforementioned "barnacle" artifacts as shown in Fig. 4 and the experiments in Sec. 5.

General "regularization" or cleanup approaches The goal of our specialized decimation is to support standard surface reconstruction procedures (of which fast implementations are widely available) and to reliably prevent the identified artifacts as efficiently as possible. Modifications to the reconstruction step itself to improve mesh quality exist [RW08, TPG99], but can generate non-manifold configurations and increase implementation complexity.

Instead, we want to briefly sketch a "mesh cleanup" approach inspired by the work of Moore and Warren [MW91, MW92] which

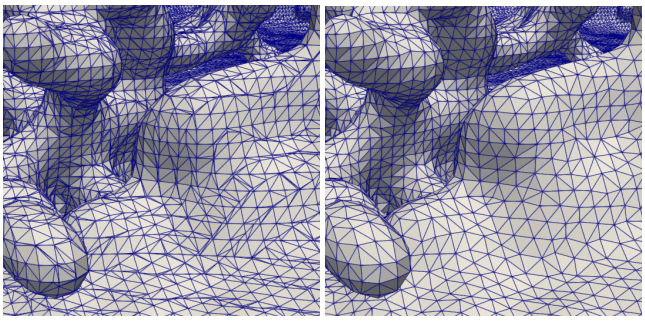


Figure 6: *Marching cubes surface mesh before (left) and after (right) applying the outlined "mesh cleanup" procedure which successfully removes typical sliver triangles.*

improves mesh quality by displacement or snapping of vertices while still being easy to implement as a post-processing step. Adapted for our use, it can be summarized as: for all mesh vertices perform incremental half-edge collapses with all one-ring neighbors sharing the same closest MC grid point and displace target vertices to the average position of the collapsed vertices. Optionally, this can be restricted to vertices within a margin around a MC grid point. An example of applying this procedure is shown in Fig. 6.

Similar to our specialized decimation approach, this cleanup step prevents "barnacle" artifacts from occurring during smoothing. Additionally, it reduces the total number of vertices and triangles for a typical surface mesh by around 30% without significantly affecting surface quality, especially when followed by smoothing. However, this also makes it slower than our specialized approach because it performs many more collapse operations: in our experiments it was 3-5x slower than our specialized decimation. Depending on the application, either of the presented methods might be preferable.

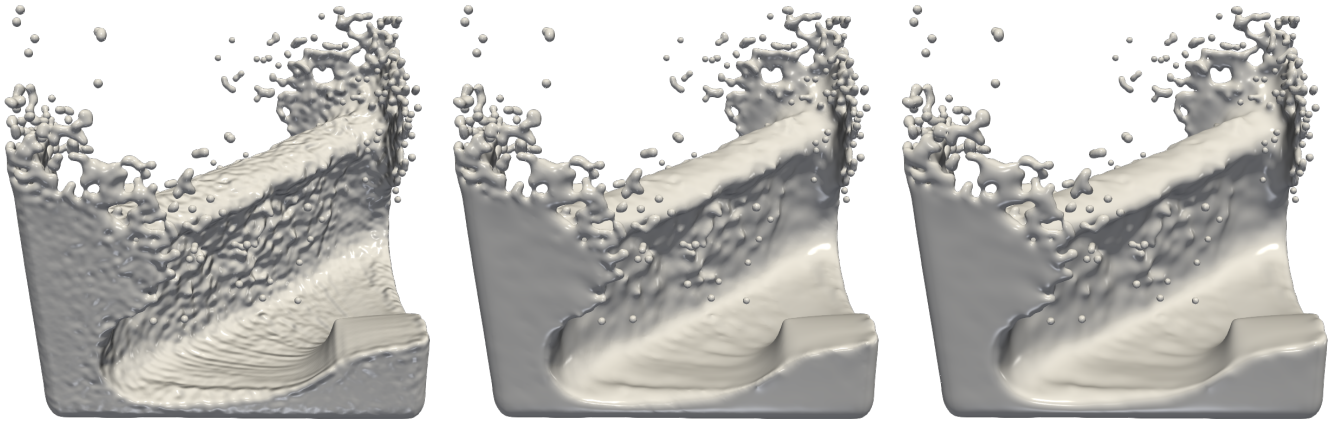
3.4. Smoothing of Vertex Normals

The primary purpose of the meshes treated by our pipeline is rendering to visualize the fluid. Therefore, as a last step in our post-processing pipeline, we consider the mesh normals to improve the perceived smoothness in renderings. For surface reconstruction of SPH simulation data, it is common to evaluate an SPH approximation of the normals (i.e., normalized gradient of the color field) on the mesh [MCG03] as this leads to smoother results than computing normals from the mesh. However, this approach is not well suited for meshes that are already smoothed with the weighted Laplacian smoothing that we propose. Evaluating the SPH normals on the smoothed mesh has the opposite effect and makes the mesh appear bumpier than the geometry really is as shown in Fig. 7a.

Instead, we recommend computing vertex normals as an area or angle weighted average of incident faces (see [BKP*10]) followed by optional normal smoothing iterations. Inspired by Laplacian smoothing we compute the updated normal of a vertex i as

$$\mathbf{n}_i^{n+1} = \frac{\sum_{j \in \mathcal{V}_i} \mathbf{n}_j^n}{\| \sum_{j \in \mathcal{V}_i} \mathbf{n}_j^n \|}. \tag{12}$$

This noticeably improves the smoothness of the rendered surface as shown in Fig. 7c.



(a) *SPH normals* **(b)** *Geometry normals* **(c)** *Smoothed geometry normals*

Figure 7: *Using the normalized SPH gradient of the color field as normals for rendering of the mesh reintroduces bumps that were previously smoothed out (a). Area weighted face normals are a better choice (b). With additional smoothing (here 10 iterations) of the geometry-based normals, we can visually reduce remaining bumpiness, especially at "corners" or "edges" of the fluid near the boundary (c).*

4. Implementation

For our implementation of the proposed procedure, we treated MC as a black box and implemented the smoothing pipeline purely as a post-processing step. We first have to perform a neighborhood search in order to evaluate the per-particle weighted neighbor counts as given by Eq. (11). For the neighborhood search, we used a variant of spatial hashing due to its ease of implementation and decent performance [THM*03, IABT10]. However, to improve performance further, especially for larger amounts of particles, one could use an octree-accelerated approach with vectorization [FFWL*22]. To interpolate the per-particle values to the mesh, we have to identify the particles influencing a given vertex. This could be done by performing a special neighborhood search that treats mesh vertices as a separate set of particles (e.g., like boundary particles in standard SPH). For ease of implementation, we instead chose to use an off-the-shelf implementation of the R*-tree spatial acceleration structure [BKSS90] which allows to query all particles within kernel support radius of a mesh vertex.

While this allows us to implement the method in a relatively small amount of code, performance can be improved significantly by integrating it into a MC surface reconstruction implementation as we can re-use parts of the pipeline. A neighborhood search has to be performed for the reconstruction anyway if the level-set is defined using an SPH sum. Furthermore, the interpolation of the per-particle values to the mesh vertices can be performed analogously to the interpolation of any fluid properties or attributes to the surface (like e.g. the velocity field) which is a feature that is generally useful outside of our specific application. Therefore, two relatively expensive steps of our proposed method are basically "free" when integrating it with the surface reconstruction which significantly reduces the cost of computing the smoothing weights in particular. The only remaining noteworthy performance overhead on top of a standard MC surface reconstruction are the smoothing iterations themselves (which are quite cheap) and our decimation procedure.

5. Results

We implemented our smoothing approach in the open-source surface reconstruction tool "splashsurf" [L*23]. To demonstrate our method on realistic simulation data, we perform simulations with the SPH framework "SPlisHSPlasH" [B*23] using DF-SPH [BK17], volume maps boundary handling [BKWK19] and a micropolar turbulence model [BKKW17].

Double dam break with dragons The first example shows a simple double dam break, colliding with a cubic static boundary and four static dragon geometries. This scene, depicted in Fig. 8, shows a lot of splashing and isolated particles, that are captured very well and not being shrunk by our smoothing procedure. By design, there is hardly any visible difference in the splashes between the smoothed and unsmoothed versions.

On top of the bulk of the fluid however, it is clear that our smoothing has the greatest effect. In the smoothed version, it is possible to see intricate details of the surface being retained, while the unsmoothed version is very bumpy. This is particularly emphasized around the specular highlights. Here, the smoothed version shows smooth deviations in the surface due to shallow waves being propagated, while the unsmoothed version makes the surface look very "rough". For more details, we refer to the supplemental video.

The mesh shown in Fig. 8 has about 2.5 million vertices and 5 million triangles. On average per frame, the entire surface reconstruction with smoothing took 5.2s. Out of this, marching cubes and file IO took on average 2.5s (48%). From our post-processing pipeline 0.72s (13.8%) were spent on mesh decimation, 0.81s (15.6%) on the weight computation and interpolation, 0.2s (3.8%) on mesh smoothing and 0.22s (4.2%) on normal computation and smoothing.

Dynamic objects In the second scene, we show the effectiveness of our surface smoothing approach in a simulation of fluid interaction with rigid bodies. The scene contains a beach ball, a duckling

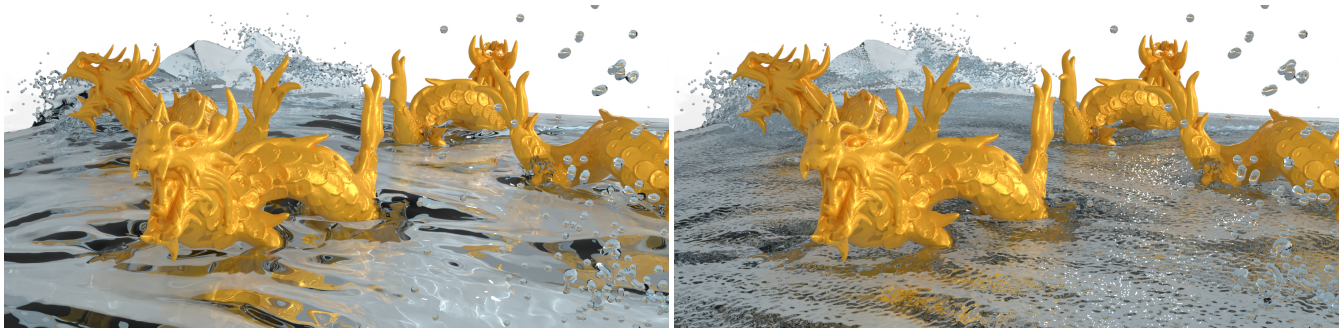


Figure 8: *A double dam break collides with four static dragon models (2.7 million fluid particles). The images show the surface reconstruction with our proposed smoothing (left) and surface reconstruction without smoothing (right).*

and a walrus of varying densities. The objects are dropped into a container into which fluid is poured from two rectangular emitters.

A still frame of this scene is shown in Fig. 1, where we compare the surface reconstruction using our proposed smoothing, against the original unsmoothed MC reconstruction. Our method is able to retain all of the small-scale detail in splashes and isolated particles, as well as overall shape and motion of the bulk fluid. In addition, in the bulk of the fluid, our approach is able to entirely smooth out the small-scale oscillations caused by the particle arrangement. For more details, we refer to the supplemental video.

The mesh shown in Fig. 1 has about 1.2 million vertices and 2.4 million triangles. On average per frame, the entire surface reconstruction with smoothing took 2.59s. Out of this, marching cubes and file IO took on average 1.34s (52%). From our post-processing pipeline 0.43s (16.6%) were spent on mesh decimation, 0.22s (8.5%) on the weight computation and interpolation, 0.11s (4.2%) on mesh smoothing and 0.10s (3.9%) on normal computation and smoothing.

6. Conclusion

We have presented a new post-processing pipeline for surface meshes of particle-based fluids generated using marching cubes. The core component of this pipeline is Laplacian smoothing with our proposed feature weights based on the neighborhood deficiency of particles. This pipeline is able to remove bumps caused by irregular placed particles, producing smooth and flat surfaces without losing more intricate features, like drops, splashes or sharp edges where the fluid meets a boundary. Additionally, we showed how artifacts caused by low-quality MC triangulations can be detected and removed with only small overhead. To further improve the final visuals when using physically-based rendering, we showed how a smoothed vertex normal field can be computed. Our pipeline is efficient, parallelizes well and introduces only few parameters.

The main limitations of our approach come from the inherent properties of Laplacian smoothing. Significantly more smoothing iterations are required when using higher MC resolutions, as the smoothing scales with edge length while the scale of bumps stays constant. We recommend keeping the cube sizes just small enough, that isolated particles are still captured appropriately.

In the future, it could be explored if directly smoothing the scalar values at the MC grid points using our positional weights yields good results. The main benefit of this would be better topology, as triangles are directly generated for the smooth case instead of capturing the bumps. It would also be interesting to further explore the different introduced strategies to avoid artifacts due to bad quality marching cubes triangles, e.g., to improve performance of the mesh-cleanup procedure.

References

[AAIT12] AKINCI G., AKINCI N., IHMSEN M., TESCHNER M.: An efficient surface reconstruction pipeline for particle-based fluids. In *Workshop on Virtual Reality Interaction and Physical Simulation* (2012). 2

[AIAT12] AKINCI G., IHMSEN M., AKINCI N., TESCHNER M.: Parallel surface reconstruction for particle-based fluids. *Computer Graphics Forum 31*, 6 (2012). 2

[Aki14] AKINCI G.: *Efficient surface reconstruction for SPH fluids*. PhD thesis, Albert-Ludwigs-Universität Freiburg, 2014. 3

[APKG07] ADAMS B., PAULY M., KEISER R., GUIBAS L. J.: Adaptively sampled particle fluids. *ACM Transactions on Graphics 26*, 3 (2007). 2

[B*23] BENDER J., ET AL.: "SPlisHSPlasH" Library. https://splishsplash.physics-simulation.org, 2023. 7

[BGB15] BHATTACHARYA H., GAO Y., BARGTEIL A. W.: A level-set method for skinning animated particle data. *IEEE Transactions on Visualization and Computer Graphics 21*, 3 (2015). 2

[BK17] BENDER J., KOSCHIER D.: Divergence-free SPH for incompressible and viscous fluids. *Transactions on Visualization and Computer Graphics 23*, 3 (2017). 1, 7

[BKKW17] BENDER J., KOSCHIER D., KUGELSTADT T., WEILER M.: A micropolar material model for turbulent SPH fluids. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2017). 7

[BKP*10] BOTSCH M., KOBBELT L., PAULY M., ALLIEZ P., LEVY B.: *Polygon Mesh Processing*. CRC Press, 2010. 3, 5, 6

[BKSS90] BECKMANN N., KRIEGEL H.-P., SCHNEIDER R., SEEGER B.: The r*-tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD International Conference on Management of Data* (1990). 7

[BKWK19] BENDER J., KUGELSTADT T., WEILER M., KOSCHIER D.: Volume maps: An implicit boundary representation for SPH. In *Proceedings of ACM SIGGRAPH Conference on Motion, Interaction and Games* (2019). 7

[Bli82] BLINN J. F.: A generalization of algebraic surface drawing. *ACM Transactions on Graphics 1*, 3 (1982). 2

[Blo88] BLOOMENTHAL J.: Polygonization of implicit surfaces. *Computer Aided Geometric Design 5*, 4 (1988). 2

[dALJ*15] DE ARAÚJO B. R., LOPES D. S., JEPP P., JORGE J. A., WYVILL B.: A survey on implicit surface polygonization. *ACM Computing Surveys 47*, 4 (2015). 3

[DK91] DOI A., KOIDE A.: An efficient method of triangulating equivalued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems 74*, 1 (1991). 2

[DMSB99] DESBRUN M., MEYER M., SCHRÖDER P., BARR A. H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In *ACM Conference on Computer Graphics and Interactive Techniques* (1999). 3

[FFWL*22] FERNÁNDEZ-FERNÁNDEZ J. A., WESTHOFEN L., LÖSCHNER F., JESKE S. R., LONGVA A., BENDER J.: Fast octree neighborhood search for SPH simulations. *ACM Transactions on Graphics 41*, 6 (2022). 7

[GH95] GUEZIEC A., HUMMEL R.: Exploiting triangulated surface extraction using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics 1*, 4 (1995). 2

[GHB*20] GISSLER C., HENNE A., BAND S., PEER A., TESCHNER M.: An implicit compressible SPH solver for snow simulation. *ACM Transactions on Graphics* (2020). 1

[Gib98] GIBSON S. F. F.: Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of the 1998 IEEE Symposium on Volume Visualization* (1998). 2

[GM77] GINGOLD R. A., MONAGHAN J.: Smoothed Particle Hydrodynamics: Theory and Application to Non-Spherical Stars. *Monthly Notices of the Royal Astronomical Society*, 181 (1977). 1

[IABT10] IHMSEN M., AKINCI N., BECKER M., TESCHNER M.: A parallel SPH implementation on multi-core CPUs. *Computer Graphics Forum 30*, 1 (2010). 7

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. *ACM Transactions on Graphics 21*, 3 (2002). 2

[KBFF*21] KUGELSTADT T., BENDER J., FERNÁNDEZ-FERNÁNDEZ J. A., JESKE S. R., LÖSCHNER F., LONGVA A.: Fast corotated elastic SPH solids with implicit zero-energy mode control. *Proc. ACM Comput. Graph. Interact. Tech. 4*, 3 (2021). 1

[KBST22] KOSCHIER D., BENDER J., SOLENTHALER B., TESCHNER M.: A survey on SPH methods in computer graphics. *Computer Graphics Forum 41*, 2 (2022). 1

[KCVS98] KOBBELT L., CAMPAGNA S., VORSATZ J., SEIDEL H.-P.: Interactive multi-resolution modeling on arbitrary meshes. In *ACM Conference on Computer Graphics and Interactive Techniques* (1998). 5

[L*23] LÖSCHNER F., ET AL.: "splashsurf" Surface Reconstruction Software. https://splashsurf.physics-simulation.org, 2023. 7

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Conference on Computer Graphics and Interactive Techniques* (1987). 2, 3

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003). 2, 6

[MSD07] MÜLLER M., SCHIRM S., DUTHALER S.: Screen space meshes. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007). 2

[MW91] MOORE D., WARREN J.: *Mesh Displacement: An Improved Contouring Method for Trivariate Data*. techreport TR-91-166, Rice Univesity, Department of Computer Science, 1991. 2, 6

[MW92] MOORE D., WARREN J.: *Compact Isocontours from Sampled Data*. Academic Press Professional, Inc., USA, 1992, p. 23–28. 2, 6

[NISA06] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Laplacian mesh optimization. In *International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia* (2006). 3, 4

[RW08] RAMAN S., WENGER R.: Quality isosurface mesh generation using an extended marching cubes lookup table. *Computer Graphics Forum 27*, 3 (2008). 2, 6

[Tau95] TAUBIN G.: Curve and surface smoothing without shrinkage. In *Proceedings of IEEE International Conference on Computer Vision* (1995). 2, 3

[Tau01] TAUBIN G.: Linear anisotropic mesh filters. *RC22213 Computer Science* (2001). 3

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANTES D., GROSS M. H.: Optimized spatial hashing for collision detection of deformable objects. In *International Symposium on Vision, Modeling, and Visualization* (2003). 7

[TPG99] TREECE G., PRAGER R., GEE A.: Regularised marching tetrahedra: improved iso-surface extraction. *Computers & Graphics 23*, 4 (1999). 6

[VMM99] VOLLMER J., MENCL R., MULLER H.: Improved laplacian smoothing of noisy surface meshes. *Computer Graphics Forum 18*, 3 (1999). 2, 3

[Wil08] WILLIAMS B. W.: *Fluid surface reconstruction from particles*. Master's thesis, University of British Columbia, 2008. 2

[YT13] YU J., TURK G.: Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Transactions on Graphics 32*, 1 (2013). 2

[YWTY12] YU J., WOJTAN C., TURK G., YAP C.: Explicit mesh surfaces for particle based fluids. *Computer Graphics Forum 31*, 2 (2012). 2

[ZB05] ZHU Y., BRIDSON R.: Animating sand as a fluid. *ACM Transactions on Graphics 24*, 3 (2005). 2