# The Beta Mesh: a New Approach for Temporally Coherent Particle Skinning *

Hagit Schechter†
University of British Columbia

Robert Bridson‡
University of British Columbia

## Abstract

We present a novel surface reconstruction approach for generating surfaces from animated particle data, targeting temporally coherent surface reconstruction that can also approximate smooth surfaces and capture fine details. Our beta mesh algorithm uses the union of balls as a building block to reach temporal coherence. First we construct mesh vertices from sphere intersection points, and declare faces on the spheres surface guided by connectivity intelligence derived from the alpha mesh. Then we smooth the beta vertices positions to reflect smooth surfaces, and subdivide the mesh using weighted centroids. We also highlight the strengths and weaknesses of the related alpha mesh for animation purposes, and discuss ways of leveraging its qualities. Open issues are discussed to outline what is still lacking in order to make our algorithm a ready-to-use surfacing technique. Nevertheless, we advocate using the beta mesh approach in future surface reconstruction research to benefit from its unique properties.

**Keywords:** Temporal coherence, particle skinning, surface reconstruction, alpha shapes, union of balls, particle simulation, lagrangian simulation, fluid simulation, surface smoothing.

## 1 Introduction

Fully three-dimensional liquid animation using physics-based simulation is now common in feature film, and is becoming attractive for interactive applications. We focus on the geometric problem of tracking, capturing, or reconstructing the surface of the liquid throughout a free-surface liquid simulation.

We present exploratory work in the area of surface reconstruction from particle fluids, both identifying an issue (temporal coherence) which has yet to be thoroughly addressed, and proposing new algorithms which bring us closer to a solution. We assembled a convincing algorithm for 2D, and made progress in extending it to 3D, while at the same time realizing new ideas will still be required to fully address the challenges in 3D. In particular, we mean by "temporal coherence" that we want the geometry to change continuously in time w.r.t. the motion of the underlying particles, even if the topology of its tessellation may discretely change arbitrarily. The challenge we directly address is producing continuously varying geometry despite the underlying topology used in the construction algorithm changing abruptly in time.

Two classes of surfacing methods dominate the field of fully three-dimensional free-surface liquid animation: implicit Eulerian methods and particle-based mesh-free methods. In the latter, also known as Lagrangian methods, the liquid volume is sampled by particles which are advected in space by the governing equations. From the beginnings of liquid animation, these methods have been popular due to various advantages such as inherent mass conservation and ease of implementation. Note that we exclude heightfield methods, very commonly used for ocean surfaces among other things, as being 2.5 dimensional – here displacements applied to an underlying surface mesh already work extremely well.

Particles fill the liquid volume roughly uniformly, generally with some level of irregular sampling depending on the underlying simulation algorithm. The particles move by physical laws and hence follow smooth trajectories (up to discretization limitations), can form varying topologies through splitting and merging, and evolve into smooth surfaces as well as fine features of various kinds.

SPH is a popular Lagrangian approach successfully used in Computer Graphics to simulate a wide range of phenomena and applications, such as free surface flow [Müller et al. 2003], two-phase flow [Müller et al. 2005; Solenthaler and Pajarola 2008], bubbles [Hong et al. 2008], porous flow [Lenaerts et al. 2008], and even viscoplastic solids [Desbrun and Cani 1996; Becker et al. 2009]. FLIP has also been a popular choice in recent years for simulating different phenomena such as water and sand [Zhu and Bridson 2005], smoke [Schechter and Bridson 2008], and multi-phase flow [Boyd and Bridson 2012]. Particle-based methods such as these do not guarantee homogenous distribution among particles. In typical "weakly compressible" SPH the stiffness of the Equation of State relating pressure to density determines the level of compressibility, which is reflected in sampling variations, as well as small-scale irregularities inherent in the kernel-based smoothing approach and sometimes artifacts such as clustering. FLIP enforces the fluid velocity incompressibility on the scale of a background grid, but lacks a built-in mechanism to control how particles are spread in the domain at a sub-grid resolution; FLIP is also robust to quite irregular distributions which is sometimes exploited deliberately by increasing particle sampling in a thin shell next to the surface. Moreover, in both of these techniques, once the liquid starts splashing and forming thin features, or interacts with solid walls, the particles are likely to be unevenly distributed in the domain.

The last step in such a simulation pipeline is rendering the liquid surface reconstructed from particles to form a realistic animation. Creating a surface representation from the particles is necessary to use standard rendering techniques. Moreover, representing the surface as a mesh, that presumably forms a tessellation of a smooth limit surface, is more amenable to further geometric processing, rendering with displacement textures, specular reflection and refraction, motion blur, etc.

The crucial surface characteristics we aim for can be classified in two groups. The first family of properties is related to the fluid geometry at an instant in time. We desire smooth liquid surfaces (particularly apparent in specular reflection or refraction), as well as the maintenance of interesting details, such as ripples, droplets, tendrils, and thin sheets. These properties have been the focus of sustained research throughout the years, and is relatively well-explored even if still under-appreciated in the academic graphics research community. The second family of properties relates to the temporal coherence of the animated surface. We want the surface to evolve smoothly in time while performing natural fluid merges and splits, and avoiding any pops or jitters that reflect surfacing limitations. Opposed to the spatial surface properties, the temporal property has had very little attention in related research, and is the primary focus of our work.

We here propose a surfacing technique dubbed the "beta mesh", for

*UBC Computer Science Technical Report TR-2013-02

†e-mail: hagitsch@cs.ubc.ca

‡e-mail: rbridson@cs.ubc.ca

skinning particles in a temporally coherent manner, while attempting to capture smooth surfaces and fine features. At present it does not quite achieve all goals, but we believe it introduces some techniques worthy of further study and shows promise in many common scenarios.

Two prior surfacing techniques which inspired our work are essential building blocks. The *union of balls* introduced in the early stages of computer graphics research, is totally faithful to particle positions, and hence evolves smoothly in time. However only in an infinitesimal limit it can capture the flatness of the surface, and in a standard simulation flow the surface will have spherical bumps. The *alpha shape* [Edelsbrunner et al. 1983; Edelsbrunner and Mücke 1994] is a dual form of the union of balls [Edelsbrunner 1993], can be directly converted to a mesh form, the *alpha mesh*, by keeping vertices and connectivity, and replacing arcs with edges. The alpha mesh can reproduce flat surfaces despite irregularity in particle sampling, and in fact no other method known to us can do it similarly. On the other hand, the alpha mesh lacks temporal coherence as we will explain later: moving a set of vertices in the mesh by an arbitrarily small amount can cause an edge flip in a non-planar region, thin features can connect and disconnect from one step to the next, and new vertices can appear or disappear from the mesh in a nonsmooth manner.

The beta mesh algorithm attempts to benefit from the best properties of these two techniques. While we define it in terms of the union of balls, and thus inherit many of the good properties of the union of balls, our algorithm for construction is based on the alpha mesh, making use of duality. We also include a temporally-coherent smoothing post-processing operation which admits a temporally-coherent and spatially smooth limit surface, potentially of use to other constructions.

Our beta mesh technique is not fully resolved yet. While in 2D it clearly demonstrates the desired properties, our 3D implementation still requires further research in order to meet our goals. In 3D the beta mesh has a lot more topological problems. The bigger challenge is the definition of beta faces: while it is unambiguous in 2D and moreover provides a unique transversal on the alpha mesh vertices, in 3D ambiguity is present, especially on thin sheets, surface holes, and their stitching to the fluid volumetric parts. The construction can reveal beta faces that are not connected at all, or are not connected in straightforward way, which makes the mesh design very challenging. This also leads to difficulties in post-processing tasks such as expanding the beta mesh volume to include all the alpha mesh vertices. In addition, thin features such as tendrils, represented in the alpha mesh as degenerate edges (i.e. with no triangles), do not have a representation in the beta mesh in its current form.

## 2 Related Work

Since the introduction of particle systems, direct particle rendering has always been a popular choice for certain applications [Reeves 1983; Sims 1990]. In the context of liquids, this is appropriate for spray or foam, but for smoother, more coherent liquid surfaces — especially with reflection and refraction effects — this is a nontrivial undertaking.

The *union of balls* algorithm wraps each particle in a sphere of a given radius. Given smooth particle trajectories, the outer surface of the union of balls also changes smoothly in time as it is entirely faithful to particle positions. However by construction it cannot represent flat surfaces other than in the theoretical limit of infinitesimal particle spacing, and with nonzero spacing the surface suffers from kinks (discontinuous normals) at the intersection between overlapping spheres.

To the best of our knowledge the union of balls representation has not been directly used for rendering, however it is often used as a guide in various contexts, e.g. cell fraction computation in MAC grids for accurate pressure solves [Batty 2010], clipping a Voronoi diagram of the particles [Sin et al. 2009], and determining the signed distance field in a one phase flow [Foster and Fedkiw 2001] and a two phase flow [Boyd and Bridson 2012].

The foundational algorithm for smooth surface reconstruction from particles was introduced by Blinn [Blinn 1982]. A summation of Gaussian density distribution functions is used, resulting in a surface that is often called "metaballs", or "blobbies", since it generalizes balls. It too is temporally coherent, but while the surface is smooth (in fact, with Gaussians, it is infinitely differentiable) it has a hard time simultaneously approximating flat surfaces and thin features. Typically there is no middle ground between bumpy "cottage cheese"-like reconstruction and overly "blobby" but smooth reconstruction which loses features. Blinn's approach has been the basis for various later works which attempt to overcome this difficulty.

Edelsbrunner et al. [1983] introduced the *alpha shape* in 2D, which was extended to 3D later on [Edelsbrunner and Mücke 1994]. Its associated mesh captures flat surfaces exceptionally well, and provides an extremely sharp and detailed reconstruction of tendrils and thin features. However, it does not transition in time smoothly, which is critical for surfacing simulation data. Edelsbrunner et al. [Edelsbrunner 1993] demonstrated its properties as the dual form of the union of balls, which is of particular interest for us, raising the question of how the good properties of the union of balls and the alpha mesh can be merged in a single algorithmic framework.

Solving for an implicit levelset function without re-initialization has been used in [Premože et al. 2003] to reconstruct the surface. Avoiding levelset re-initialization helps to get smooth transitions, however may deviate in time from the real surface due to numerical errors. Moreover the surfacing computation time in this method was significantly more than the actual simulation time, which is undesirable, and constituted a time-dependent problem in itself ruling out the possibility of parallel reconstruction across all frames independently.

Several works in the past decade focused on improving the surface smoothness in direct surface reconstruction from particles. Müeller et al. [Müller et al. 2003] defined the surface as an isosurface of a scalar field that is smoothed using an isotropic kernel. Zhu and Bridson [Zhu and Bridson 2005] used an implicit function of distance to a sphere, replacing the actual position and radius with a computed weighted sum position of nearby particle centers, and a weighted sum radius of particle radii. The distance function is then sampled on a grid, a smoothing pass over the grid is performed, and an isosurface is then extracted from the grid. Their results are considerably smoother than the classic blobby spheres surface when extracting detailed features. Adams et al. [Adams et al. 2007] refined the algorithm by tracking the particle-to-surface distances over time, and using it in the weighted distance to sphere equation. They experimented with their technique on adaptively-sized particle radii, as well as on homogenous radii. Further improvement was introduced by Yu and Turk [Yu and Turk 2010]. They detect the anisotropy direction of particle positions by performing a Principal Component Analysis over the covariance matrix of particle neighborhood. Then they perform a smoothing step that repositions the centers of these smoothing kernels. Compensating for the anisotropy in the flow significantly reduces the surface bumpiness. The above direct surfacing from particles improve the reconstructed surface smoothness, however none of them enforces or handles the temporal coherence requirement.

An alternative approach for surface reconstruction was proposed

by Williams [Williams 2008]. A nonlinear optimization problem is solved iteratively to achieve optimal smoothness for the surface mesh, while constrained to lie between two union of balls surfaces of different radii centered at the particles. Bhatacharya et al. [Bhatacharya et al. 2011] introduced a variant of this technique using level sets rather than meshes. Perfectly flat surfaces can be generated using these methods under certain conditions, however some drawbacks are present. First, small changes in particle positions may lead to quite large surface changes since the optimization is global and only geared towards surface flatness. As a result there is no guarantee on reaching smooth transitions of the surface in time.

Another group of works deals with surface reconstruction from a point cloud, e.g. Alexa et al. [Alexa et al. 2001]. We believe that these works face different challenges than ours. While they deal with denoising point positions in order to construct proper geometry, our objective is in some sense opposite. In our case the particle positions are the ground truth, and our goal is to stick to it while constructing a surface surrounding them in the best way and conforming to certain requirements. On the other hand, some tools and techniques are common to the two classes of surfacing problems. We would like to highlight three works from this group that are of a particular relevance to our work. Bernardini et al. [Bernardini et al. 1999] build a manifold subset of the alpha shape similar to our alpha mesh construction, but they do it incrementally. Amenta et al. [Amenta et al. 2001] approximate the medial axis and its inverse and then compute the surface that lies between them. The power diagram used in their construction is closely related to the alpha mesh that we use. Tam and Heidrich [Tam and Heidrich 2004] compute the dual shape of the weighted Delaunay triangulation, extract singular points, and triangulate guided by the dual shape triangles. The Delaunay dual shape is in fact the alpha mesh, which drives the triangulation of both their technique and ours. Nevertheless several major differences are present. First, our main purpose is processing data originating from an actual particle simulation, and addressing the temporal coherence requirement. This poses new challenges in our case, particularly in the interpretation of thin sheets of different kinds, and dealing with unique geometrical layouts of fine simulation features that are not likely to be present in a given well defined geometry. We cannot fill in gaps in the particle data to compensate for poor distribution sections, and our algorithm needs to handle such segments robustly. In addition we aim producing a smooth limit surface that approximates the liquid's natural smoothness, and yet providing a proper geometrical representation for degenerate alpha simplices.

# 3 The Algorithm

## 3.1 Overview

We assume we are in $d$-dimensional Euclidean space: points can be drawn from $\mathbb{R}^d$, and the distance between points is the usual Euclidean distance. We are most interested in $d = 3$, but the case of $d = 2$ is also instructive.

A *closed ball* of radius $r$ centered on a point $p$ is the set of points within distance $r$ of $p$: $\{x : \|x - p\| \leq r\}$.

The *open ball* is where the distance is strictly less than $r$: $\{x : \|x - p\| < r\}$.

The open ball is the *interior* of the closed ball.

The points at distance exactly $r$ from $p$ form the *boundary* of the ball (whether open or closed).

For the following we will assume a set of $n$ special points $\{x_i\}_{i=1}^n$ which we will call *particles*. For our application these will in fact be fluid particles from a simulation.

An *empty ball* is an open ball with no particles in its interior.

The *union of balls* of the particles is the union of $n$ closed balls centered on each of the particles. It defines a surface around the particles that is temporally coherent, but cannot characterize flat surfaces. The balls have the same radius $r$, which is fixed ahead of time and is correlated with the particles typical spacing $dx$.

The algorithm proceeds in three steps: the alpha mesh construction described in Section 3.2, the alpha graph processing described in Section 3.3, and the beta mesh construction described in Section 3.4. Each of these sections includes an overview/definitions subsection that provides mathematical definitions and algorithm overview, and a detailed algorithm subsection that discusses the practical implementation in 3D, provides 2D/3D illustrations, and outlines a pseudocode of the algorithm or parts of it.

## 3.2 Alpha Mesh

### 3.2.1 Alpha Mesh - Overview and Definitions

The *alpha shape* for a set of vertices $S = \{x_i\}_{i=1}^n$ is a subset of vertices and arcs connecting them, s.t. every arc is part of a closed ball that contains $d$ vertices of $S$ and its open ball form is an empty ball. It may be easier to think of it as the complement of the union of all empty balls.

We say a particle is an *alpha vertex* if it appears on the boundary of the alpha shape.

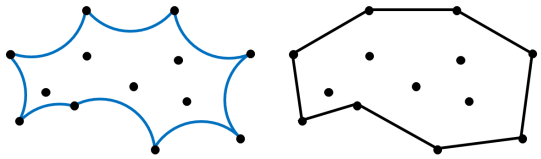
We will from now on assume the alpha vertices are in "general position". In particular, we will assume:

- no two alpha vertices are coincident,
- no more than $d$ may lie on the boundary of an empty ball where $d$ is the dimension of the space, and
- for each set of $d$ alpha vertices, if there is any ball whose boundary contains them then the center of the ball cannot be on the (hyper-)plane through the alpha vertices.

The first and last assumptions imply that if $d$ alpha vertices appear on the boundary of a ball, they appear on exactly two such balls — one on each side of the plane through the alpha vertices, corresponding to the two possible normal vectors for the plane. Using some variation of the right-hand-rule, we can similarly associate each of the two balls with one of the two orientations of the $d$ alpha vertices.

The *alpha mesh* is an oriented mesh formed on the alpha vertices, with simplex facets (triangles in 3D) corresponding to the ball-shaped sections on the boundary of the alpha shape. More precisely, an oriented facet of $d$ alpha vertices appears in the alpha mesh if they appear on the boundary of an empty ball, and in fact the unique ball associated with the oriented facet as above is empty. The normal of the face points towards the center of that empty ball, or in other words is an outward-pointing normal. See Figure 1 for an illustration of the alpha shape and the alpha mesh, or [Edelsbrunner et al. 1983] for more illustrations.

Note that there may be *degenerate* simplices in the alpha mesh that are not part of any facet: maximal collections of $d - 1$ or less vertices which appear on the boundary of an empty ball. These are not naturally oriented, or can be thought of as appearing with all possible orientations. An *isolated vertex* is not connected to any other vertex. In particular, every closed ball containing the vertex



(a) (b)

**Figure 1:** ***Alpha Shape and Alpha Mesh.*** *Particles are shown in black dots. (a) Alpha shape (blue). (b) Alpha mesh (back): alpha shape arcs were converted to edges.*

is empty of other particles. An *isolated edge* in the alpha mesh is a pair of alpha vertices distance $r$ or less apart for which every ball containing them is empty of all other particles.

#### 3.2.2 Alpha Mesh - Algorithm Details

Our algorithm runs through all the particles in the set. For for every particle and for every ordered pair in the set within specified distance from the particle, it examines the circumscribing sphere of the ordered triplet. If no other particle in the set is contained in the sphere it declares the ordered triplet as a triangle in the mesh. A pseudocode for the alpha mesh construction in 3D is given in Algorithm 1, and the circumscribing sphere criterion is described in Algorithm 2.

**Algorithm 1** Alpha mesh construction

**Input:** set of particle positions $P = \{\mathbf{p}_i\}$, radius $r$
**Output:** set of triangles $T = \{t_i\}$

**for all** $\mathbf{p}_i \in P$ **do**
  let $N_i = \{\mathbf{p}'_i\}$ s.t. $\mathbf{p}'_i \neq \mathbf{p}_i,\ dist(\mathbf{p}'_i, \mathbf{p}_i) <= 2r$
  **for all** pair $\mathbf{p}_j, \mathbf{p}_k \in N_i$ s.t. $\mathbf{p}_j \neq \mathbf{p}_k$ **do**
    $(succeeded, \mathbf{c})$ = $build\_sphere(< \mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k >, r)$
    **if** succeeded **then**
      **if** $dist(\mathbf{p}, \mathbf{c}) > r\ \forall \mathbf{p} \in N_i \setminus \{\mathbf{p}_j, \mathbf{p}_k\}$ **then**
        $t =< \mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k >$
        $T \leftarrow T \cup \{t\}$

**Algorithm 2** Build circumscribing sphere

**Input:** triplet $< \mathbf{x0}, \mathbf{x1}, \mathbf{x2} >$, radius $r$
**Output:** flag *succeeded*, center $\mathbf{c}$

$\mathbf{n} = (\mathbf{x0} - \mathbf{x1}) \times (\mathbf{x1} - \mathbf{x2})$
$radius_x = \frac{dist(\mathbf{x0},\mathbf{x1})dist(\mathbf{x1},\mathbf{x2})dist(\mathbf{x2},\mathbf{x0})}{2\|(\mathbf{n})\|}$
**if** $radius_x > r$ **then**
  return **false**
**else**
  $\alpha = \frac{dist(\mathbf{x1},\mathbf{x2})^2((\mathbf{x0}-\mathbf{x1})\cdot(\mathbf{x0}-\mathbf{x2}))}{2\|\mathbf{n}\|^2}$
  $\beta = \frac{dist(\mathbf{x0},\mathbf{x2})^2((\mathbf{x1}-\mathbf{x0})\cdot(\mathbf{x1}-\mathbf{x2}))}{2\|\mathbf{n}\|^2}$
  $\gamma = \frac{dist(\mathbf{x0},\mathbf{x1})^2((\mathbf{x2}-\mathbf{x0})\cdot(\mathbf{x2}-\mathbf{x1}))}{2\|\mathbf{n}\|^2}$
  $\mathbf{l} = \alpha\mathbf{x0} + \beta\mathbf{x1} + \gamma\mathbf{x2}$
  $t = \sqrt{(radius_x{}^2 - r^2)/\|\mathbf{n}\|^2}$
  $\mathbf{c} = \mathbf{l} + t\mathbf{n}$
  return **true**

Our algorithm does not rely on a Delaunay triangulation preprocess like Edelsbrunner's original alpha mesh algorithm [Edelsbrunner et al. 1983; Edelsbrunner and Mücke 1994]. This leads to several advantages:

- Our algorithm runs in $O(n)$ computation time where $n$ is the number of particles, assuming reasonably well-distributed particles. The overhead of the full Delaunay construction is avoided, which takes $O(n \log(n))$ or even worse with 3D thin tendrils particle data.
- Our method is trivial to parallelize: any triple can be tested independently of the other triples.
- We find the degenerate parts of the alpha mesh that are missed in the obvious Delaunay construction, and hence can directly address them.

The choice of the search radius has a critical impact on the resulting alpha mesh, as was illustrated in Edelsbrunner's work [Edelsbrunner and Mücke 1994]. A value that is too small may fail connecting relevant geometry parts, a value that is too big may wrongly inflate concave surface areas. Here it also influences the alpha graph and beta mesh derived from the alpha mesh, see the following Section 3.3 and Section 3.4. In our experiments we used $r = 1.5dx$ where $dx$ is the typical particle spacing, however more tests are required to reach best utilization of our mesh construction.

The local nature of our algorithm however causes sensitivity to rounding errors that may result in an invalid mesh, which is eliminated in the original Delaunay-driven alpha mesh construction. An illustrating example is 4 particles that form a square of $dx$ edge length. All or none of the triangles may be added to the alpha mesh, instead of adding precisely two triangles that share a diagonal. Another example is a very dense cluster of particles: our construction may miss a dense triplet triangulation, causing an opening in the cluster's outer surface. To avoid such problems we validate the alpha mesh using a simple edge count test: every oriented edge in the mesh must have a pair edge in an opposite direction. If the check fails we perturb the particle positions by a small tolerance and re-run the alpha mesh construction. On the vast majority of the frames that we processed a single alpha mesh pass was used, and only a few required re-running the algorithm once more.

We call attention to several special cases in the alpha mesh structure. A thin layer of fluid that forms a fairly flat sheet of particles, namely a *thin sheet*, is a common scenario that causes degeneracy in the mesh. The alpha mesh representation of a thin sheet in 3D is a double-sided sheet of triangles, clamping the fluid particles on both sides. Each oriented triangle in the thin sheet has an associated triangle with the opposite orientation: this pair forms a *double-sided triangle*. We highlight that the alpha mesh representation of thin sheets is unique, due to the thin and fairly flat surface that is formed. In contrast the union of balls technique produces a bumpy surface in this case. Double-sided triangles may be present in other scenarios as well, such as when shared between two tangent fluid bubbles. The 3D *degenerate simplices* that are not included in the alpha mesh are isolated particles and isolated edges; currently we collect those particles into a set of external particles and render them as spheres.

### 3.3 Alpha Graph

#### 3.3.1 Alpha Graph - Overview and Definitions

We can impose a manifold-like adjacency structure on the $d$-simplex faces of the alpha mesh. Consider two faces with $d-1$ alpha vertices in common, and look at all faces incident on that $d-1$ simplex: they may be sorted radially by angle around the common $d-1$ simplex. The two faces are adjacent if no other of the co-incident faces appears radially between them, where between is defined as the outward radial section or between where their normals point.

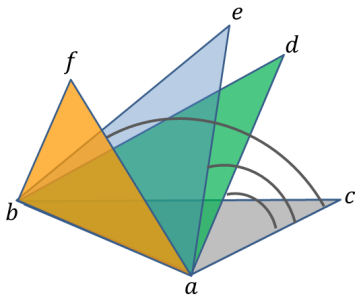



**Figure 2:** ***Triangles Order in Alpha Face Construction.*** *The order between triangles that share an edge is determined by the signed volume of the tetrahedra, or equivalently by the dihedral angle between two triangles. This sorting guarantees that faces do not intersect.*

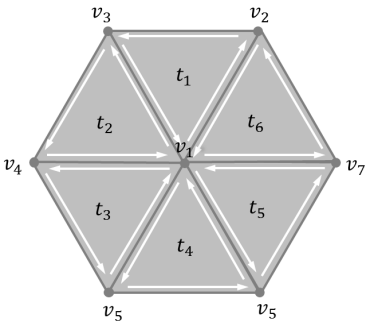



**Figure 3:** ***Alpha Graph Example.*** *An illustration of the alpha graph of a thin sheet. Alpha mesh is presented in gray triangles/dots, triangle orientation is highlighted in white.* $v_1$ *participates in two faces: the front face and the back face, both consist of six triangles. The front face around* $v_1$ *is the order set* $\{t_1, t_2, t_3, t_4, t_5, t_6\}$*; it is a closed face. The faces around vertices* $v_2$-$v_7$ *are open faces.* $v_2$ *for example participates in two faces: one on each side, each consists of two triangles, the front face is the ordered set* $\{t_1, t_6\}$.

We structure the alpha graph information in *graph faces*, each composed of an alpha vertex and an ordered set of $d$-simplices. The graph faces do not intersect each other by the construction described above, other than when two simplex boundaries overlap. An alpha vertex can impose more than one graph face.

While the non-degenerate parts of the alpha mesh (the $d$-simplex faces) form a closed and oriented sub-mesh, it might well not be manifold. More than two faces may be incident on a common $d-1$ simplex.

Note that although the above definitions may stand for $d > 3$, handling the graph connectivity in high dimensions is quite complicated; we focus here on our primary interest where $2 \leq d \leq 3$ and neglect higher dimensions.

#### 3.3.2 Alpha Graph - Algorithm Details

The algorithm starts by assigning for each alpha vertex $v_i$ its adjacency set, i.e. the alpha triangles that it participates in. We then divide those triangles into faces: disjoint groups of ordered triangle. We define an order among the vertex triangles: $t_j$ follows $t_i$ w.r.t. $v_i$ if they share an edge in opposite direction that includes $v_i$. Double-sided triangles make an exception: a triangle cannot follow its other side pair triangle. This definition is sufficient to impose a unique order when there is only one triangle that follows each triangle in the adjacency set, however often this is not the case. When two triangles $t_j$ and $t_k$ follow triangle $t_i$, we want to connect them by the dihedral angle they form with $t_i$. We can avoid calculating the dihedral angles and instead use a robust signed volume predicate: if the signed volume of the tetrahedron (a),(b),(d),(e) in Figure 2 is positive it implies the dihedral angle to the triangle with (e) is larger; if negative the dihedral angle to the triangle with (d) is larger. This comparison operator is then all the sorting algorithm needs to find the correct dihedral angle order. A *closed face* is when the first triangle in the ordered set follows the last triangle. On the rim of a thin sheet each *boundary face* forms an *open face*, i.e. the last triangle in the ordered set does not have a next triangle. Note that thin sheet boundary faces in 2D are processed seamlessly: the opposite side edge can be considered as a possible next edge, and if no other edge takes precedence it means that the thin sheet boundary has reached and traversal continues on the opposite side. In contrast, the 3D boundary faces on both sides are treated as open faces and the graph construction algorithm identifies the start/end triangles when processing the face. See Figure 3 for an illustration of a thin sheet and its alpha faces. A pseudocode for the alpha graph algorithm is provided in Algorithm 3.

**Algorithm 3** Alpha graph

**Input:** alpha vertices $P = \{p_i\}$, alpha triangles $T = \{t_i\}$
**Output:** $\forall p \in P$, faces of p $\{f_p^k\}_k$

**for all** $p \in P$ **do**
  let $T_p = \{t_i\}$ be the set of triangles that include $p$
  **for all** $t_i \in T_p$ **do**
    **if** $t_i$ is processed **continue**
    let $T_p^i \subseteq T_p$ be the set of triangles that follow $t_i$
    sort $T_p^i$ by signed volume w.r.t. $t_i$
    $t_{curr} \leftarrow t_j$, $t_{first} \leftarrow t_j$
    **while** true **do**
      mark $t_{curr}$ as processed
      find the first available triangle $t_j \in T_p^i$
      declare: $t_j$ follows $t_{curr}$
      **if** $t_{first}$ follows $t_j$ **then**
        declare: face $f$ complete
        **break**
      $t_{curr} \leftarrow t_j$
      $T_p^i \leftarrow T_p^i \setminus \{t_j\}$

An illustrative example for an alpha vertex that participates in more than one face is when a fluid bubble is tangent to the fluid surface. If the touching area between the bubble and the air surface include a single vertex, every triangle in the vertex adjacency list has a single triangle that follows it without requiring for dihedral angle sorting. If the touching area is composed of more than one vertex, the dihedral angle sorting determines how to arrange the triangles in faces. In both cases every shared vertex has two faces, one for the air surface and the other for the bubble surface.

### 3.4 Beta Mesh

#### 3.4.1 Beta Mesh - Overview and Definitions

The beta mesh is loosely the dual of the alpha mesh of a set of points.

A *pure beta vertex* can be defined in two equivalent ways. In $d$ dimensions, it is any intersection of the boundaries of $d$ different balls centered on the particles, which is not inside any open ball

centered on a particle — think of it as a point on the boundary of the union-of-balls where $d$ balls meet. Such a point is equivalently the center of a ball whose boundary contains $d$ particles and whose interior is empty of particles. From this it is clear a beta vertex is "dual" to a facet from the alpha mesh. We use the preface "pure" for this type of vertices to distinguish it from other types of beta vertices that will later be defined.

The *pure beta mesh* is an oriented mesh formed on the pure beta vertices, with general faces corresponding to the connected ball-shaped parts of the union-of-balls boundary. These faces may have an arbitrary number of sides, and are often not planar — think of them more as a topological construction rather than well-defined geometry.

For each oriented $d$-simplex in the alpha mesh, a pure beta vertex can be constructed as the center of the unique corresponding empty ball. Each edge of the beta mesh corresponds precisely to a pair of adjacent alpha mesh faces with no intervening alpha mesh faces between them (going angularly around the common subsimplex). Given the alpha graph structure described above, the beta mesh can be built quite easily: there is a one-to-one correspondence between graph faces and beta faces. Alpha vertices and beta faces are also tightly coupled, yet it is not a one-to-one relationship. Multiple beta faces can share the same alpha vertex – when the alpha vertex has more than one graph face. In this case there is a one-to-many correspondence between alpha vertices and beta faces. At the same time, degenerate parts of the alpha mesh such as isolated particles or degenerate edges have no associated beta vertices or faces.

In 2D the pure beta mesh is a well defined mesh. However in 3D it is not a feasible geometrical representation, and we aim turning it into a triangular mesh, to easily visualize the surface as well as apply advanced rendering tasks such as motion blur. To turn the pure beta mesh into a triangular mesh, we define a new type of beta vertex, the *alphabeta vertex*, each corresponds to a graph face in a one-to-one relationship. We initially position the alphabeta vertices at the projection of the alpha vertex to the approximated beta face plane, and triangulate them with the pure beta vertices of the face.

The initial positions of beta vertices however does not yield the desired smooth surface. Beta vertex positions are prone to particle distribution fluctuations, reflected in slight alpha triangulation variation that get amplified when converted into beta positions. The alphabeta vertices positioned on the faces planes do not respect the curvature of curved surfaces. Aiming at a smooth surface we hence run a smoothing step. First, we smooth each pure beta vertex by computing a weighted average of all the pure beta vertices in adjacent faces: the center point of every face edge is weighted by the edge length. After updating all the pure beta vertices, we similarly reposition each alphabeta vertex using a weighted average of all the smoothed pure beta vertices in its face. Note that boundary beta vertices do not go through the smoothing procedure to prevent them from getting pushed to the thin sheet interior.

The beta mesh may have degeneracies related to degeneracies in the alpha mesh — or rather, missing features. An isolated vertex in the alpha mesh corresponds to a ball in the union-of-balls which doesn't intersect any other ball, and thus has no associated beta mesh vertices: it vanishes from the beta mesh. In general an isolated $d-1$ or less simplex in the alpha mesh does not contain any proper beta mesh vertices since they are formed from the intersection of $d$ balls, and thus these features are missing from the beta mesh. In order to faithfully capture the motion of the liquid and present splashes and drops that separate and merge into the fluid geometry, we define a third type of beta vertex: the *external beta vertex*. These vertices do not take part in the beta mesh, and instead are rendered as spheres.

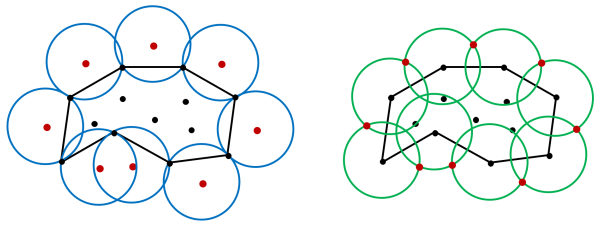



**Figure 5:** ***Construction of Pure Beta Vertices.*** *Pure beta vertices can be computed equivalently from the alpha shape disk centers, or from the union of balls intersections. Particles are shown in black. (a) Circumscribing circles (blue) associated with the directed alpha mesh edges (black) define the position of pure beta vertices (red) as the circle center. (b) Union of balls (green) define the position of pure beta vertices (red) as the intersection between two or more circles.*

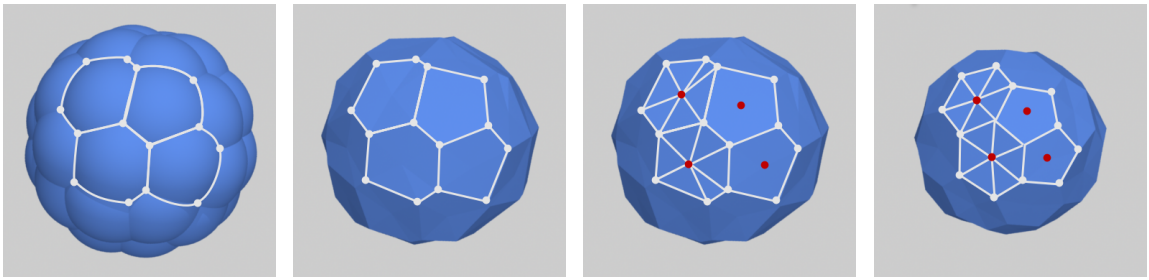
#### 3.4.2 Beta Mesh - Algorithm Details

This section describes the beta mesh construction process. Figure 4 provides a 3D visualization of the algorithm running on a sphere. The algorithm proceeds in several steps, exemplified for 3D in Figure 4 and for 2D in Figure 8.

**Step 1: beta vertices.** Pure beta vertices are created: a single vertex for each alpha triangle, initially located at the center of the alpha circumscribing sphere. See Figure 5 for an illustration of the dual meaning of pure beta vertices. Together with the alpha graph connectivity the pure beta mesh is now defined, using pure beta vertices and pure beta faces. To be able to triangulate the beta faces late on, we create alphabeta vertices: a single vertex for each alpha graph face, initially positioned on the approximated plane passing through the face, projecting the alpha vertex to the plane. Computing the plane normal uses the weighted normals of adjacent alpha face edges. Last, external beta vertices are created from isolated alpha vertices and degenerated alpha mesh edges. They do not take part in the beta mesh, and are saved for the frame visualization step.

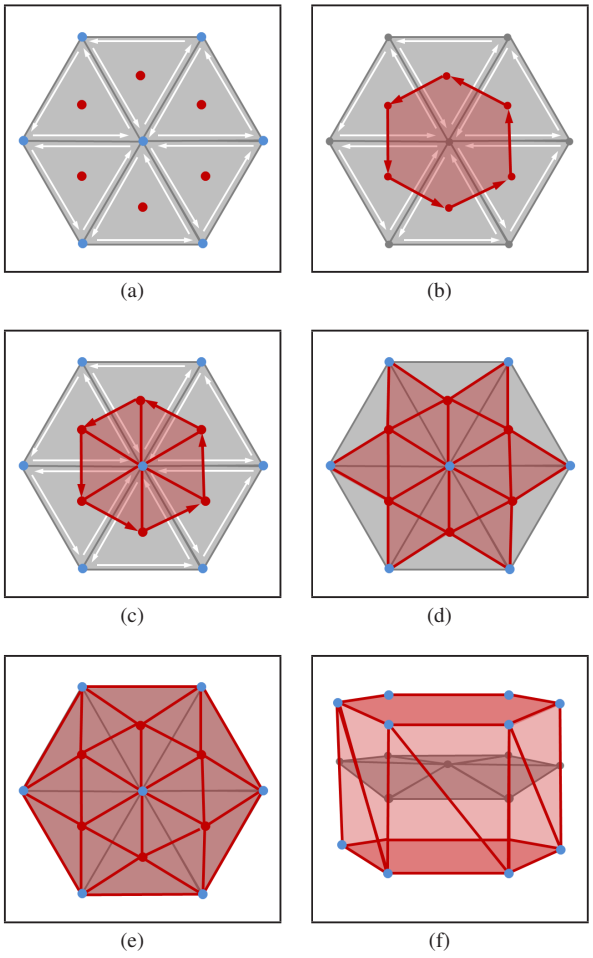
**Step 2: tessellation.** The tessellation process relies on the alpha graph: each closed alpha face defined a closed beta face with the same number of triangles. Each open alpha face with $k$ triangles defines an open beta face with $k-1$ triangles. To close the rim of thin sheets we define new triangles that do not correlate with alpha mesh triangles. The tessellation process proceeds in several steps. First we tessellate beta faces, starting with closed faces. Every alphabeta vertex connects with the pure beta vertices in its face, using the orientation defined in the alpha graph. Similarity boundary face triangulation is performed. All the triangles created so far are composed of one alphabeta vertex and two pure beta vertices. Then we complete the tessellation of the thin sheet rim. First we add a triangle connecting the two alphabeta vertices with the pure beta vertex on each side of the boundary edge. Finally we close the rim by tessellating quads of alphabeta vertices at the boundary edge with two triangles. Thin sheets in the alpha mesh result in flat sheets in the beta mesh of $2r$ thickness. See Figure 6 for an illustration of a thin sheet tessellation process.

**Step 3: smoothing.** The smoothing process repositions the beta vertices but leaves the connectivity unchanged. First, the position of every pure beta vertex is recomputed as a weighted average of all the pure beta vertices in its faces. Then, the position of every alphabeta vertex is recomputed as a weighted average of the pure beta vertices in its face. A pseudocode is provided in Algorithm



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

**Figure 4:** ***Beta Mesh Algorithm Visualization (3D).*** *(a) Starting with the "union of balls" surface, initial pure beta vertices (white dots) are positioned at the intersections of three or more balls. Pure beta faces, derived from the alpha graph, can be visualized as ball caps bounded by arcs of ball intersections. (b) Arcs are replaced by edges; note that face vertices are typically not coplanar. (c) Alphabeta vertices (red dots) initial position is set and face triangulation is performed. (d) The smoothing process repositions pure beta vertices at the weighted sum of (initial) pure beta vertices of adjacent faces, and alphabeta vertices at the weighted sum of the face (new) pure beta vertices. The smoothing result is presented, demonstrating improved mesh quality.*



(a) (b) (c) (d) (e) (f)

**Figure 6:** ***Beta Tessellation of a Thin Sheet.*** *Presentation of alpha mesh/vertices (gray), beta mesh (red), pure beta vertices (red dots), alphabeta vertices (blue dots); top view is used except for (f) with side view. (a) Pure beta and alphabeta vertices. (b) Pure beta face (without alphabeta vertices). (c) Tessellation of the center face. (d) Tessellation of the boundary faces. (e) Complete the front/back tessellation. (f) Close the rim (side view).*

**Algorithm 4** Beta smoothing

**Input:** beta vertices $P = \{\mathbf{p}_i\}$, beta triangles $T = \{t_i\}$, beta faces $G = \{f_i\}$
**Output:** new beta positions $P' = \{\mathbf{p}'_i\}$

**for all** $\mathbf{p} \in P$ pure beta vertex **do**
  let $G^p = \{f_j\}$ be the set of faces of $\mathbf{p}$
  $\mathbf{p_{new}} \leftarrow \mathbf{0}$, $weight \leftarrow 0$
  **for all** $f_j \in G^p$ **do**
    **for all** $< \mathbf{p}_a, \mathbf{p}_b > \in f_j$ pure beta edge **do**
      $w \leftarrow dist(\mathbf{p}_a, \mathbf{p}_b)$
      $\mathbf{p}' \leftarrow \frac{\mathbf{p}_a + \mathbf{p}_b}{2}$
      $weight \leftarrow weight + w$
      $\mathbf{p_{new}} \leftarrow \mathbf{p_{new}} + w\mathbf{p}'$
  $\mathbf{p_{new}} \leftarrow \frac{\mathbf{p_{new}}}{weight}$
**for all** $\mathbf{p} \in P$ pure beta vertex **do**
  $\mathbf{p} \leftarrow \mathbf{p}_{new}$
**for all** $\mathbf{q} \in P$ alphabeta vertex **do**
  let $f$ be the beta face of $\mathbf{q}$
  $\mathbf{q_{new}} \leftarrow \mathbf{0}$, $weight \leftarrow 0$
  **for all** $< q_a, q_b > \in f$ pure beta edge **do**
    $w \leftarrow dist(\mathbf{q}_a, \mathbf{q}_b)$
    $\mathbf{q}' \leftarrow \frac{\mathbf{q}_a + \mathbf{q}_b}{2}$
    $weight \leftarrow weight + w$
    $\mathbf{q_{new}} \leftarrow \mathbf{q_{new}} + w\mathbf{q}'$
  $\mathbf{q_{new}} \leftarrow \frac{\mathbf{q_{new}}}{weight}$
  $\mathbf{q} \leftarrow \mathbf{q_{new}}$

4. Figure 7 demonstrates the improved mesh quality following the smoothing process.

**Step 4: Resolving Beta/Alpha Mesh Intersections (optional).** We propose detecting where the beta mesh intersects the alpha mesh and preventing those intersections, otherwise a particle could lie outside of the beta mesh. The algorithm goes through all the alpha faces, and checks if there is a triangle that intersects a beta triangle that belongs to the correlated beta face. If an intersection occurs it pushes the alphabeta vertex to the alpha vertex position. By construction a pure beta vertex cannot cause such intersections, hence the alphabeta vertex position is the one that needs to be adjusted.

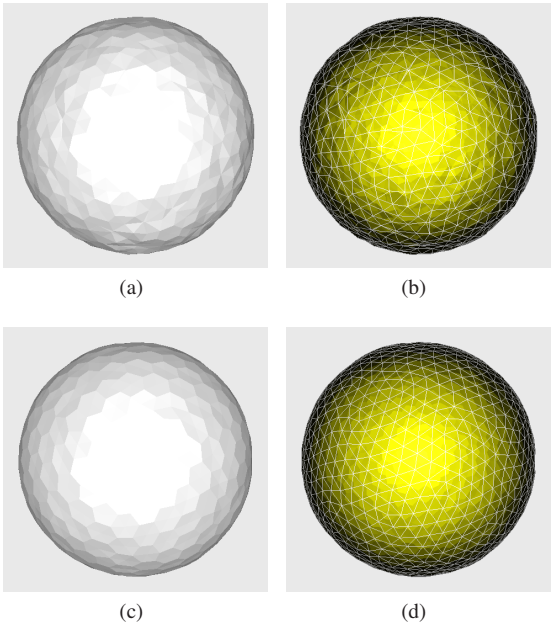


**Figure 7:** ***Beta Mesh Smoothing Step.*** *Top: beta mesh when faces are connected to a weighted centroid. Bottom: the result after the smoothing step. Left: standard mesh representation. Right: front mesh triangles drawn in wire mode.*

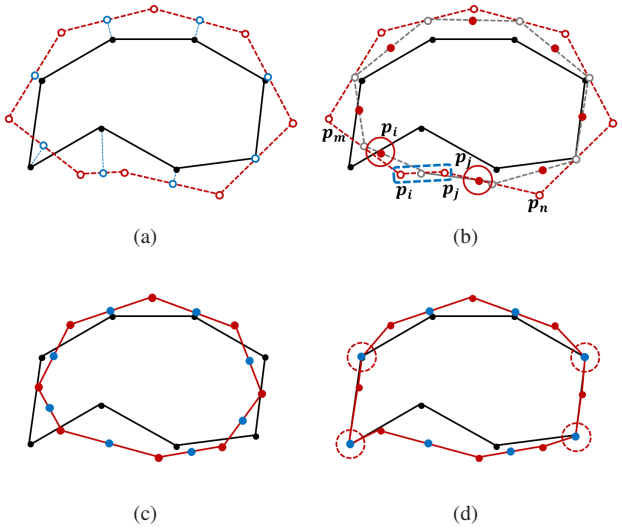



**Figure 8:** ***Beta Mesh Algorithm Illustration (2D).*** *Alpha mesh is presented in black, beta mesh is presented in red. (a) Beta mesh with initial vertex positions, showing pure beta vertices (hollow red dots), alphabeta vertices (hollow blue dots), and beta mesh edges (dotted red lines). (b) Beta mesh smoothing first step: reposition pure beta vertices. Initial mesh is presented in hollow red dots and dotted red lines, alphabeta vertices are not presented. Face centroids (hollow gray dots) are used to compute the new pure beta vertices (solid red dots). Note: the smoothing process fixes short edges, see the initial edge $< p_i, p_j >$ highlighted in blue vs. the distance between the final $p_i, p_j$ vertices highlighted in red circles. (c) Beta mesh after smoothing, showing edges (red lines), pure beta vertices (solid red dots), and alphabeta vertices (solid blue dots). (d) Beta mesh after resolving alpha/beta mesh intersections. Alphabeta vertices that were repositioned are highlighted.*

### 3.5 Temporal Coherence

The primary goal of the beta mesh surfacing algorithm is to maintain temporal coherence and yet achieve all the other desirable geometrical properties. Its construction relies both on the union of balls to benefit from its temporal coherence property, and on the alpha mesh to benefit from its geometrical characteristics. In this section we discuss the temporal coherence problems in the alpha mesh, and explain how the beta mesh construction overcomes these issues and maintains temporal coherence.

#### 3.5.1 Alpha Mesh - Temporal Coherence

The union of balls is temporally coherent, and so is the alpha shape being its dual [Edelsbrunner 1993]. However when alpha shape arcs and sphere sub-sections are replaced with alpha mesh edges and triangles, temporal coherence is lost. We identify three scenarios of temporal coherence problems in the alpha mesh.

Edge flip. The same set of alpha vertices that exists in two consecutive frames may go through a topological change, reflected in an unsmooth mesh change. Consider for example four alpha vertices that form two non-planar triangles that share an edge. Particle movement on the next step can change the triangulation of these vertices and two new non-planar triangles are created on the opposite diagonal.

Vertex appearance/disappearance. Marginal particle movement can cause a new alpha vertex to appear or an existing vertex to disappear from the mesh, resulting in a major geometrical change. E.g. an isolated particle that is just outside of the circumscribing sphere joins the alpha mesh on the next step, see illustration in Figure 9 (a) (b). Despite the very small change in the particle position, a major geometric change of magnitude $r$ in the worse case takes place in the alpha mesh.

Thin features. Alpha mesh thin features may disconnect/connect from one step to the next due to a marginal particle movement that eliminates/adds a vertex triplet from the thin triangles list. An opening in the thin sheet can then be formed in one step and close again in the next step.

While the first two scenarios create true problems in a surfacing technique, the third one potentially reflects an actual physical change (fluid separating or merging). A thin fluid layer that flows on a solid wall may open and close. A fluid drop can merge and split from the fluid bulk, and nearly pushes/pops itself with very fast surface-tension-mediated dynamics which could be acceptably approximated with a temporal discontinuity. Our attention is therefore focused more on the first two scenarios, where temporal discontinuity is unacceptable.

#### 3.5.2 Beta Mesh - Temporal Coherence

We examine next the beta mesh construction steps, and provide an informal explanation for why temporal coherence is maintained. Figure 10 demonstrates this discussion, and correlates to the numbered items below.

(a) The union of balls surface is temporally coherent.

(b) The intersections between three or more balls change smoothly in time and hence the positions of pure beta vertices are temporally coherent.

(c) Every two intersecting spheres on union of balls surface form

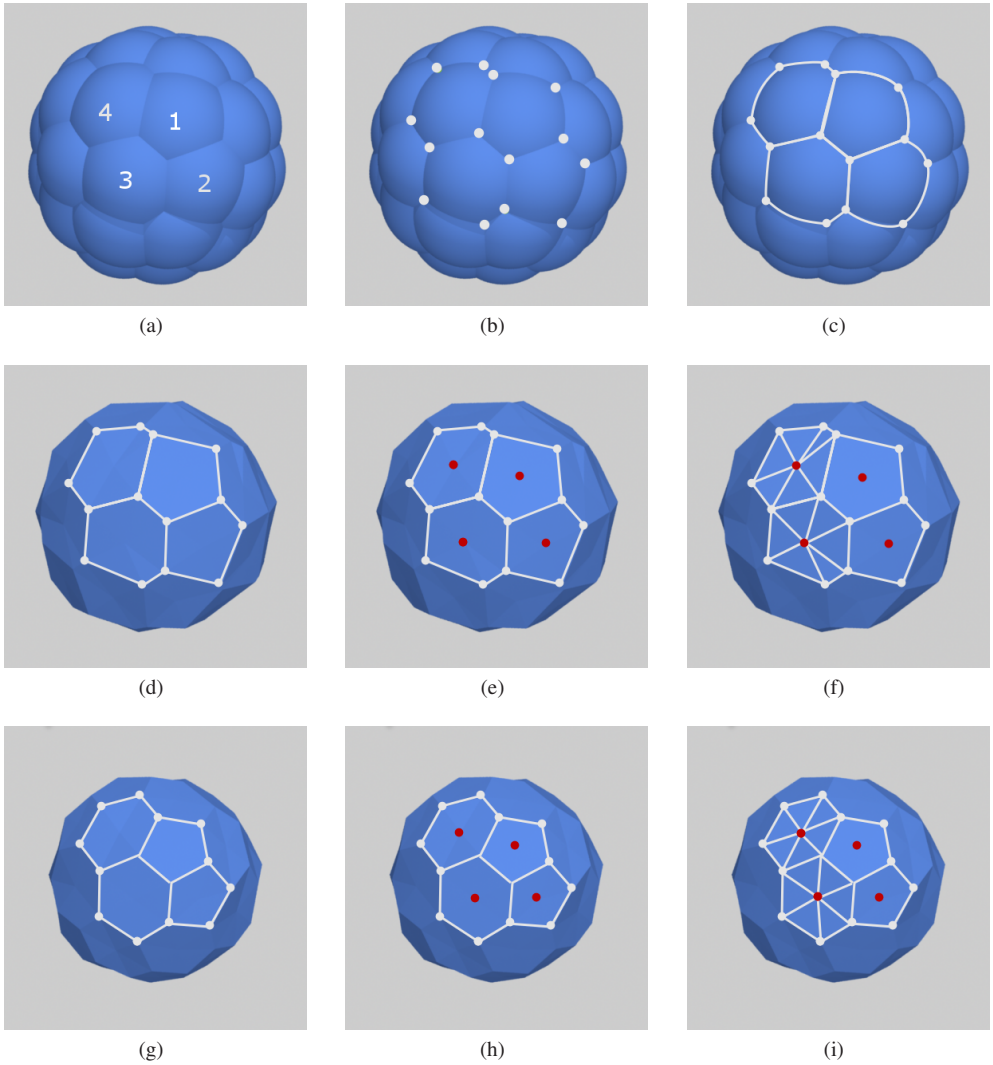



**Figure 10:** ***Temporal Coherence in the Beta Mesh Construction.*** *Upper row: union of balls. Middle row: beta mesh before smoothing. Lower row: beta mesh after smoothing. (a) Union of balls, four front balls are highlighted. (b) Pure beta vertices are created: a pure beta vertex is an intersection of three or more balls. (c) A network of arcs is marked, each arc is an intersection of two balls. (d) Arcs convert to edges, forming pure beta faces. (e) Alphabeta vertices are created on the face's approximated plane. (f) Beta faces get tessellated. (g) Beta faces after smoothing: vertices repositioned, connectivity is left unchanged. (h) Alphabeta vertices in the smoothed mesh. (i) The smoothing process improves the mesh quality, notice the new triangulation of face-3 and face-4.*

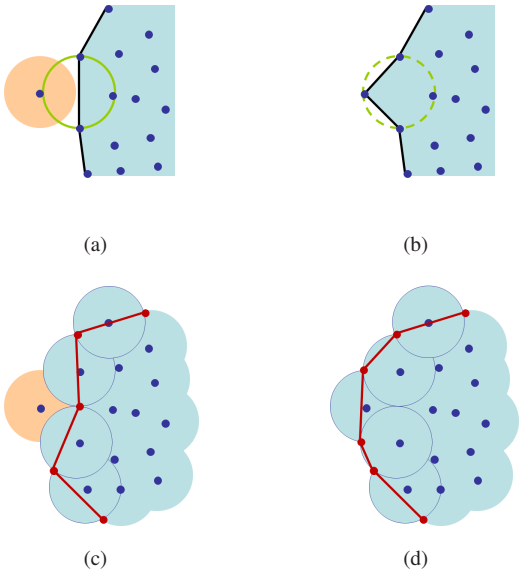



**Figure 9:** ***Water Drop Merge: Alpha Mesh vs. Beta Mesh.*** *Upper row: Alpha mesh (black line). Lower row: Beta mesh (red line). Left: Step $n$, isolated particle highlighted (orange). Right: Step $n+1$, left particle becomes part of the mesh. (a) Isolated particle is outside of the alpha circumscribing circle (green). (b) The particle enters the alpha circumscribing circle (dotted green) and becomes part of the alpha mesh, causing a sharp mesh pop. (c) Isolated particle is outside of the beta mesh. (d) The particle becomes part of the beta mesh, inducing a smoother change.*

a circular arc, or a complete circle in some cases. This network of arcs and circles changes smoothly in time.

(d) Beta faces can now be formed from a series of pure beta vertices that surround a sphere and are connected with circular arcs from the arcs network. The arcs geometry is temporally coherent as well as the induced graph/face connectivity. We can now replace the connectivity arcs between two pure beta vertices with edges, this construction still maintains temporal coherence.

(e) Alphabeta vertices are computed as a weighted sum of the face pure beta vertices, hence preserve temporal coherence.

(f) Hence triangulation of alphabeta vertices with pure beta vertices in its face also changes smoothly in time.

(g) The smoothing process recomputes pure beta vertex positions as a weighted sum of pure beta vertices in adjacent faces. The weighted sum is a temporally coherent mapping running on temporally coherent positions, therefor does not violate temporal coherence.

(h) Alphabeta vertices are recomputed as face centroids as weighted sum of the new pure beta vertices in each face; these are temporally coherent acts on temporally coherent positions, hence preserve temporal coherence.

(i) Last, triangulation of alphabeta vertices with the face pure beta vertices connects temporally coherent beta vertices and relies on a temporally coherent connectivity, hence it also changes smoothly in time.

To summarize, the beta mesh building blocks are temporally coherent, its vertices and the connectivity units are by construction temporally coherent, and therefore the resulting mesh changes smoothly in time.

Note that the circles/arcs network described above represents tendril features as a series of disconnected circles along the tendril. These circles lack intersection points, hence they do not impose pure beta vertices or beta faces. Consequently tendrils geometries (and similarly other alpha mesh degenerate features) are left outside of the beta mesh geometry in favor of keeping our temporally coherent construction rules.

Physical unsmooth changes in time are also reflected in the beta mesh, e.g. on thin sheet boundaries, or in merge/split of fluid drops or chunks. These changes however have better appearance in the beta mesh compared to the alpha mesh. Figure 9 illustrates a water drop merge, causing a sharp pop in the alpha mesh and a smoother change in the beta mesh.

## 4 Results and Discussion

Results of a 2D SPH dam break are shown in Figure 11. In our 2D implementation the graph analysis is consistent for simple volumetric regions and more complex regions with double-sided edges and thin features, and the stitching between different regions is done seamlessly. Beta mesh and alpha mesh intersections are resolved. The resulting beta mesh is smooth, and reflects alpha thin features.

Results of a 3D SPH dam break are presented in Figure 12. The top row (a)-(c) uses the union of balls particle rendering and provides the ground truth on how the surface should look like. The middle row (d)-(f) presents the alpha mesh surfacing, demonstrating an excellent representation of thin sheet in (f). Some craters are seen in the surface that may indicate high sensitivity to particle uneven distribution, or to the alpha radius chosen (here $r = 1.5dx$). In the lower row (g)-(i) the beta mesh surfacing is presented. The surfaces are much smoother than the alpha mesh surfaces. Thin sheets are mostly captured, though the boundaries implementation may not yet be fully satisfying. The craters visible in (i) reflect our major open problem, in the stitching of double-sided triangles to the fluid volumetric surface. The desired temporal coherence is achieved, demonstrated in the animations. The frame data consists of 110k particles and the surfacing run time is 19 seconds per frame that is mostly taken for the alpha mesh construction.

Figure 13 shows the surfacing of a rotating cube FLIP simulation. The frame data consists of 170k particles. Particles are unevenly distributed, which is reflected in unsmooth surfaces and long run times (up to 6 hours per frame), that again is mostly taken for alpha mesh processing. Near the surface, in this example, the average number of particles per grid cell was much higher, leading to a significantly higher constant in the $O(n)$ run time. Figures (a), (d) and (g) are most interesting for examination, demonstrating exceptionally good tendril features representation in the alpha mesh. The beta mesh is currently not able to reproduce the desired tendril look as is indicated in our open problems list. The particle inhomogeneous distribution is visible in (b), (e), (h), reflected in unsmooth surfaces in all three representations, nevertheless the beta produces much smoother surfaces compared to the alpha mesh.

### 4.1 Open Issues

The critical area to improve in our beta mesh construction is the graph interpretation of complex geometrical scenarios in the alpha mesh that may appear when double-sided alpha triangles connect to the rest of the mesh. Examples are air bubbles trapped in the fluid

resulting in holes in the surface, stitching between thin sheets and volumetric sections, or thin fluid features connecting to each other in nontrivial ways.

Another aspect that we would like to further improve is thin sheet boundary handling. Extending the surface beyond the boundary sphere center could assist towards better temporal coherence there.

Fluid tendrils are reflected in the 3D alpha mesh as degenerate edges, are presently not well captured in the beta mesh. Degenerate alpha features lend themselves to become external beta vertices, and thread-like alpha geometry may shrink its volume in the beta mesh. Better representation will increase the usability of the beta mesh technique.

Our generic smoothing procedure shows excellent results, however in some cases it may fail its designed task. A simple example is a fluid tetrahedron that will shrink after smoothing into a point. Further analysis may be required to cover such cases.

The alpha mesh procedure could also be improved. Currently our sphere criterion uses power and square root computation, which are sensitive to rounding issues. Alternative checks could be examined to avoid this sensitivity and guarantee robustness.

## 5 Conclusions

In conclusion, two surfacing techniques are presented in this work:

The alpha mesh is a well known technique, which we first use and analyze for animation purposes and provide an efficient new algorithm to compute it.

The beta mesh is a novel surfacing construction that we introduce, which blends ideas from the union of balls surfacing and the alpha mesh construction.

The alpha mesh representation of thin sheets and the tendril features are of significance for animation purposes, and no other surfacing method known to us can similarly reproduce these features. The temporal coherence issues of the alpha mesh are being considered in this work. Bringing the alpha mesh to the animation regime, and illustrating its pros and cons in this context, is of significant relevance to the field.

The beta mesh addresses an important surfacing requirement – the surface temporal coherence, which had little attention in related research works. In addition to producing smooth transitions of the surface in time, the method targets conforming to important geometrical requirements such as reflecting flat surfaces in the same algorithmic framework.

The beta mesh addresses its primary goal and its demonstrated benefits look very promising. Temporal coherence is achieved, smooth surfaces are well demonstrated, and fine features are preserved. Nevertheless, further improvements should take place to make it a useful and robust tool for animation purposes, especially on 3D complex geometrical graph sections. Handling problems and features outlined in the open issues section could make it usable for handing a broad range of phenomena.

In summary, we believe that the ideas presented in this work could serve as a basis for establishing useful particle skinning techniques in the future, that could encompass the non-trivial combination of the three desired surface requirements: temporal coherence, spatial smoothness, and fine features.

## References


ADAMS, B., PAULY, P., KEISER, R., AND GUIBAS, L. J. 2007. Adaptively sampled particle fluids. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007) 26*, 3.

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. In *Proceedings of the conference on Visualization '01*, VIS '01, 21–28.

AMENTA, N., CHOI, S., AND KOLLURI, R. K. 2001. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, SMA '01, 249–266.

BATTY, C. 2010. *Simulating Viscous Incompressible Fluids with Embedded Boundary Finite Difference Methods*. PhD thesis, The University Of British Columbia.

BECKER, M., IHMSEN, M., AND TESCHNER, M. 2009. Corotated SPH for deformable solids. In *Proceedings Eurographics Workshop on Natural Phenomena*, 27–34.

BERNARDINI, F., MITTLEMAN, J., RUSHMEIER, H., SILVA, C., AND TAUBIN, G. 1999. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 5*, 4, 349–359.

BHATACHARYA, H., GAO, Y., AND BARGTEIL, A. 2011. A level-set method for skinning animated particle data. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, 17–24.

BLINN, J. F. 1982. A generalization of algebraic surface drawing. *ACM SIGGRAPH Computer Graphics (Proceedings of SIGGRAPH 1982) 16*, 3, 273–.

BOYD, L., AND BRIDSON, R. 2012. MultiFLIP for energetic two-phase fluid simulation. *ACM Transactions on Graphics 31*, 2 (Apr.), 16:1–16:12.

DESBRUN, M., AND CANI, M.-P. 1996. Smoothed Particles: A new paradigm for animating highly deformable bodies. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, 61–76.

EDELSBRUNNER, H., AND MÜCKE, E. P. 1994. Three-dimensional alpha shapes. *ACM Transactions on Graphics 13*, 1, 43–72.

EDELSBRUNNER, H., KIRKPATRICK, D., AND SEIDEL, R. 1983. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory 29*, 4, 551–559.

EDELSBRUNNER, H. 1993. The union of balls and its dual shape. In *Proceedings of the ninth annual symposium on Computational geometry*, SCG '93, 218–231.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, 23–30.

HONG, J.-M., LEE, H.-Y., YOON, J.-C., AND KIM, C.-H. 2008. Bubbles alive. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008) 27*, 3, 48:1–48:4.

LENAERTS, T., ADAMS, B., AND DUTRÉ, P. 2008. Porous flow in particle-based fluid simulations. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008) 27*, 3, 49:1–49:8.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, 154–159.

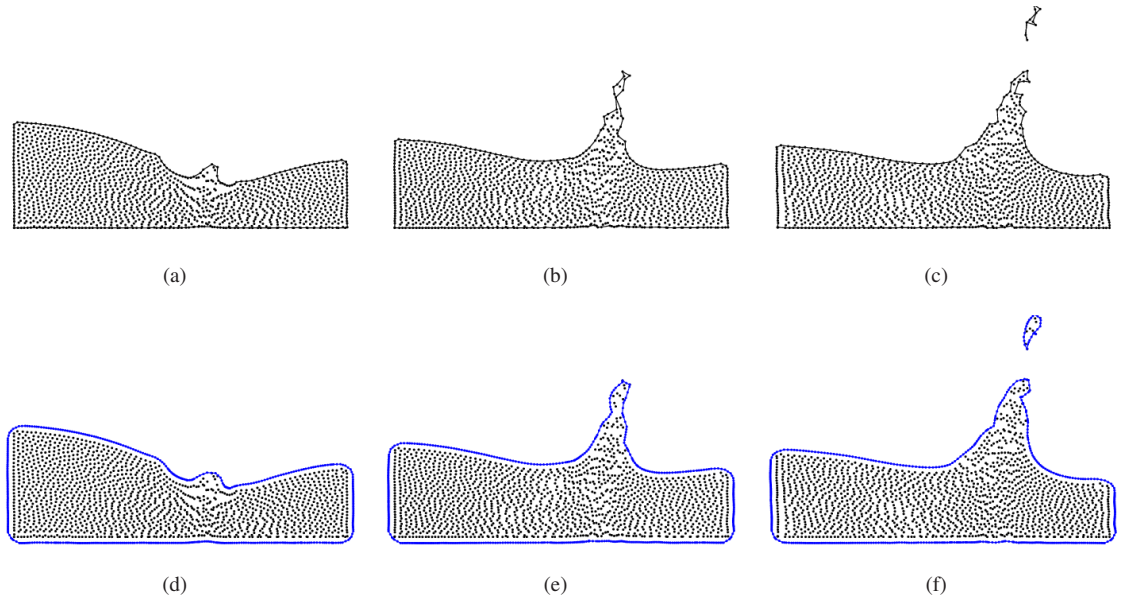



**Figure 11:** ***2D Dam Break.*** *Upper row (a)-(c): Alpha mesh. Lower row (d)-(f): Beta mesh. (equivalent frames are presented)*

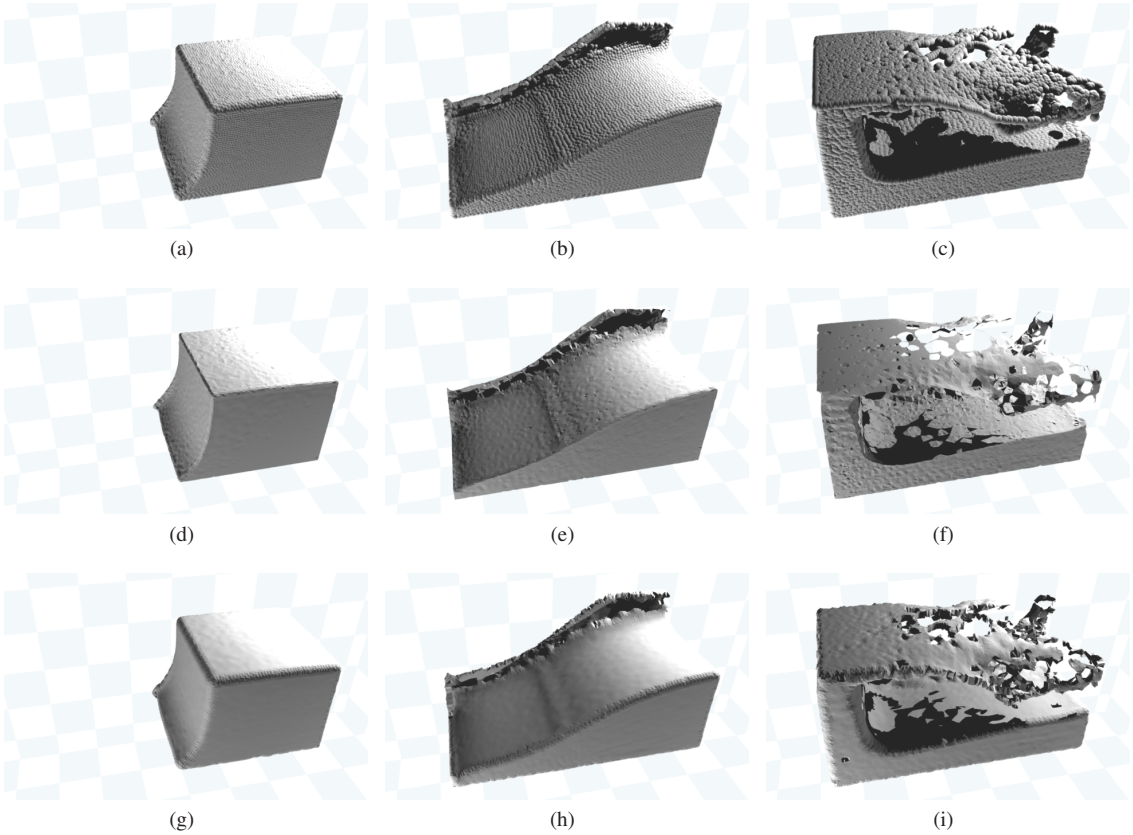



**Figure 12:** ***3D Dam Break.*** *Upper row (a)-(c): Union of Balls. Middle row (d)-(f): Alpha mesh. Lower row (g)-(i): Beta mesh. (equivalent frames are presented)*

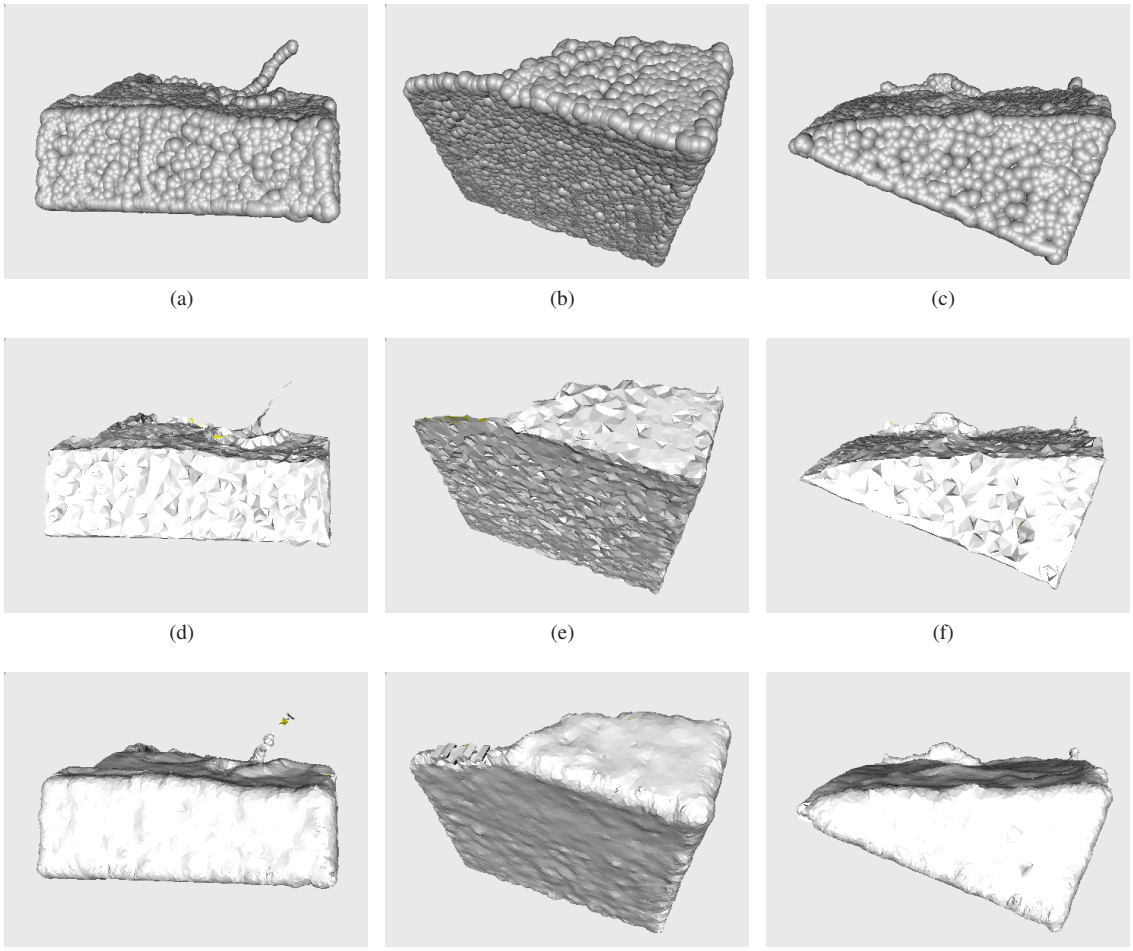



**Figure 13:** ***Rotating Cube.*** *Upper row (a)-(c): Union of Balls. Middle row (d)-(f): Alpha mesh. Lower row (g)-(i): Beta mesh. (equivalent frames are presented)*

MÜLLER, M., SOLENTHALER, B., AND KEISER, R. 2005. Particle-based fluid-fluid interaction. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, 237–244.

PREMOŽE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. T. 2003. Particle-based simulation of fluids. *Computer Graphics Forum (Proceedings of Eurographics 2003) 22*, 3, 401–410.

REEVES, W. T. 1983. Particle systems - technique for modeling a class of fuzzy objects. *ACM SIGGRAPH Computer Graphics (Proceedings of SIGGRAPH 1983) 17*, 3, 359–375.

SCHECHTER, H., AND BRIDSON, R. 2008. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, 1–7.

SIMS, K. 1990. Particle animation and rendering using data parallel computation. *ACM SIGGRAPH Computer Graphics (Proceedings of SIGGRAPH 1990) 24*, 4, 405–413.

SIN, F., BARGTEIL, A. W., AND HODGINS, J. K. 2009. A point-based method for animating incompressible flow. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, 247–255.

SOLENTHALER, B., AND PAJAROLA, R. 2008. Density contrast SPH interfaces. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 211–218.

TAM, R., AND HEIDRICH, W. 2004. Computing polygonal surfaces from unions of balls. In *Proceedings of the Computer Graphics International*, CGI '04, 86–92.

WILLIAMS, B. W. 2008. *Fluid Surface Reconstruction from Particles*. Master's thesis, The University Of British Columbia.

YU, J., AND TURK, G. 2010. Reconstructing surfaces of particle-based fluids using anisotropic kernels. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, 217–225.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 965–972.