Title
An algorithm to mesh interconnected surfaces via the Voronoi interface

Journal
Engineering with Computers, 31(1)

ISSN
0177-0667

Author
Saye, RI

Publication Date
2015

DOI
10.1007/s00366-013-0335-9

Peer reviewed

Engineering with Computers (2015) 31:123–139
DOI 10.1007/s00366-013-0335-9

ORIGINAL ARTICLE

An algorithm to mesh interconnected surfaces via the Voronoi interface

R. I. Saye

Received: 28 February 2013 / Accepted: 4 September 2013 / Published online: 6 October 2013

Abstract Many scientific and engineering problems involve interconnected surfaces meeting at junctions. For example, understanding the dynamics of a soap bubble foam can require modelling the fluid mechanics of liquid inside an intricate network of thin-film membranes. If a mesh of these surfaces is needed, the use of standard meshing algorithms often leads to voids, overlapping elements, or other artefacts near the junctions. Here, we present an algorithm to generate high-quality triangulated meshes of a set of interconnected surfaces with high surface accuracy. By capitalising on mathematical aspects of a geometric construction known as the "Voronoi interface", the algorithm first creates a topologically consistent mesh automatically, without making heuristic or complex decisions about surface topology. In particular, elements that meet at a junction do so by sharing a common edge, leading to simplifications in finite element calculations. In the second stage of the algorithm, mesh quality is improved by applying a short sequence of force-based smoothing, projection, and edge-flipping steps. Efficiency is further enhanced by using a locally adaptive time stepping scheme that prevents inversion of mesh elements, and we also comment on how the algorithm can be parallelised. Results are shown using a variety of examples arising from multiphase curvature flow, geometrically defined objects, surface reconstruction from volumetric point clouds, and a simulation of the multiscale dynamics of a cluster of soap bubbles. In this last example, generating high-quality meshes of evolving interconnected surfaces is crucial in determining thin-film liquid dynamics via finite element methods.

Keywords Interconnected surfaces · Triangulated mesh · Voronoi interface · Multiphase · Multi-material data sets

1 Introduction

Many problems in science, engineering, and medicine require meshing a network of interconnected surfaces meeting at various types of junctions. Often these meshes are needed as part of a numerical method to solve partial differential equations (PDEs) defined on the network of surfaces, such as in finite element methods. For example, modelling the fluid dynamics of the liquid contained in the thin-film membranes of a foam of soap bubbles requires solving "thin-film equations" which, in turn, are coupled at the junctions via flux boundary conditions [1]. To design accurate numerical methods for solving such PDEs, it is crucial that the generated mesh be of high quality, topologically consistent, and free of artefacts at junctions, such as voids and overlapping elements. Meshes of interconnected surfaces may also be required for purposes of visualisation, such as in examining the boundary between different connecting tissues/organs in medical imaging. Since the geometry and connectivity of the network of surfaces are often complex, meshing algorithms must also be able to reliably handle a wide variety of topological configurations.

In this paper, we present a robust and efficient mesh-generation algorithm for a network of interconnected surfaces. In the first stage of the algorithm, a topologically consistent mesh is created that has no gaps, overlaps, or

R. I. Saye (✉)
Department of Mathematics and Lawrence Berkeley National Laboratory, University of California, Berkeley, Berkeley, California 94720, USA
e-mail: saye@math.berkeley.edu

other artefacts at junctions. This step involves no ad hoc decisions about surface topology; instead, properties of the "Voronoi interface" [2] are used to guarantee consistency. The resulting mesh is suitable for many purposes, such as visualisation, but consists of many low-quality elements, so in the second stage of the algorithm a short sequence of force-based smoothing, projection, and edge-flipping iterations is applied. By using an adaptive time-stepping strategy, convergence to a high-quality mesh can be obtained within as few as 10–20 iterations, taking a fraction of a second on a typical desktop computer for a mesh with 10,000 elements.

There are many ways the set of surfaces could be defined or represented. For example, they could be explicitly parameterised using coordinate functions $(s,t)\mapsto\mathbf{X}(s,t)\in\mathbb{R}^3$. In medical applications, CT imaging and MRI produce voxel data wherein each voxel identifies different regions or types of tissue, and the boundary between the different regions defines the surfaces. In level set methods [3], moving surfaces are defined implicitly as a particular level set of an evolving scalar function. For the purposes of our discussion, we define the "interface" to be the boundary between the different regions ("phases"), i.e. the entire network of interconnected surfaces. Clearly, generating a mesh depends on how the interface is represented. In this work, our primary representation uses the "Voronoi interface" (see Sect. 3), which is a type of generalisation of the Voronoi diagram. It is used in a recently developed algorithm, the "Voronoi Implicit Interface Method" [2, 4] that tracks the interface in an evolving multiphase system. As we show below, using the Voronoi interface makes it simple to generate a high-quality mesh with elements that meet perfectly at junctions. In particular, if two or more mesh elements meet at a junction, then they will do so by sharing a common edge. This additional property ensures that the mesh has a well-defined topology, with the guarantee that mesh vertices on junctions belong to all surfaces meeting at that junction. This property leads to convenient simplifications in a finite element method when handling coupled boundary conditions at junctions. Since using the Voronoi interface makes finding a topologically consistent mesh straightforward, it may be advantageous to convert other representations (such as voxel-based data or point clouds) into this form, and we demonstrate this with an example in our results.

The outline of the paper is as follows. In the next section, we review some of the previous work on meshing multiple surfaces and compare with the approach presented here. In Sect. 3, the Voronoi interface is defined, after-which the main meshing algorithm is presented in Section 4. We then demonstrate the algorithm on various test problems in Sect. 5, showing various analyses of mesh quality and efficiency. Additional examples are also shown, including one taken from a soap bubble foam simulation in which solutions to surface PDEs computed via finite element methods is a critical component of the model.

2 Related work

Much of the previous work on meshing interconnected surfaces has focused on multi-material data sets, wherein each voxel is assigned a different label identifying different regions or materials. A variety of algorithms have been developed, which, somewhat broadly, are based on:

- *Lookup tables:* Here, a marching cubes [5] or marching tetrahedra [6–8]-style algorithm is extended to handle the case of multi-material data sets, leading to lookup tables that take into account material labels. In general, the lookup tables rapidly increase in size with the material count, and produce a mesh that has many bad-quality elements (e.g. slivers). In some cases, see e.g. [9–11], they are created through a series of heuristic decisions that decide a plausible topology, making it unclear if the resulting meshes are guaranteed to be free of artefacts. In another approach, in [12] a subdivision algorithm is used together with trilinear interpolation that leads to a mesh guaranteed to be topologically consistent; however the subdivision comes at the price of generating very large lookup tables or a large number of tiny mesh elements. In [13], a lookup table is used in the case of three materials, and in the general case, triangle removal algorithms are used, allowing meshes with boundaries and holes to be created.
- *Delaunay refinement:* These methods are an extension of typical Delaunay refinement-based algorithms to the case of multiple surfaces [14–16]. A mesh is iteratively created by adding vertices and updating mesh topology until a quality criteria is reached that terminates the algorithm. They generally involve many subtle pre-processing steps, including the need to identify and extract the triple junctions as a network of curves. The created meshes are topologically consistent, having no gaps or overlaps, and in addition, mesh elements that meet at junctions do so by sharing a common edge. Parallel implementations of these algorithms can be difficult; however, the approach has the advantage of allowing quality criteria to be defined on per-surface basis, allowing different materials to have different mesh resolutions, and volume tetrahedra meshes (consistent with the interface topology) are often produced at no extra cost.

- *Particles:* A different approach involves first placing particles (or spheres) on the set of surfaces, optimising the distribution of the particles, and then creating mesh topology through a constrained Delaunay triangulation/tetrahedrisation. A very early method using this idea was the *bubble mesh* [17]: spheres of an adaptive size are "packed" onto explicitly parameterised curves, surfaces, and volumes, in that order. In a relaxation step, the spheres are then dynamically moved according to force-based laws to optimise their distribution, before invoking a constrained Delaunay algorithm to obtain the mesh topology. More recently, in [18] an algorithm is proposed in which the particles, once seeded on the implicitly defined surfaces, are dynamically moved to optimise an energy functional that measures surface features, allowing particles to concentrate near regions of high curvature. The resulting meshes are of very high quality, but this comes at the price of a computationally expensive optimisation process; some of the meshes presented in the latter work took several hours to generate.
- *Volumetric meshing with guaranteed quality:* Another possibility is to mesh the volumes of the individual regions/phases as a tetrahedral mesh, and then extract surface meshes from the boundaries of the volumetric meshes. In the *lattice cleaving* algorithm [19] (which shares aspects with the single-region *isosurface stuffing* algorithm of Labelle and Shewchuk [20]), a body-centred cubic lattice is "cleaved", or cut, based on the approximate location of the material boundaries. In particular, vertices at the location of the cuts are "warped" by a procedure that guarantees mesh quality, such that minimum dihedral angles are bounded from below. In this method, some cases of interface topology are simplified, as only materials on the vertices of tetrahedra are used to decide topology, which can miss cases in which multiple materials interact within a tetrahedron. In another approach, Chernikov et al. [21] developed an algorithm in which the input is an image of labelled voxels, and an octree is used to output a tetrahedral volumetric mesh with bounded dihedral angles. A subdivision procedure is used near the boundary that guarantees mesh quality. However, the refinement usually produces too many elements near the boundary, and so the mesh is subsequently decimated by merging vertices while maintaining quality bounds. In addition, the created mesh does not perfectly match material boundaries, an attribute of the algorithm defended by the property that labelled voxel images do not necessarily correspond to smooth surfaces.
- *Other approaches:* In [22], a straightforward division strategy is used that subdivides the unit cube into sufficiently many smaller sub-cubes, and assigns different materials to each sub-cube. An interface between different materials is then extracted directly from the subdivided cubes, creating a topologically consistent mesh that has a staircase shape which then needs to be smoothed, affecting the accuracy of the surface representation. More recently, in [23] an octree-based approach that uses a dual contouring method is presented, also based on a subdivision algorithm that instead uses trilinear interpolation to resolve topological ambiguities. Another octree-based algorithm is developed in [24], allowing both triangular and quadrilateral surfaces meshes (as well as hexahedral volume meshes) to be created of good quality. Using instead a subdivision of a body-centred cubic lattice, in [25] a global optimisation approach is developed in which the mesh boundary is iteratively deformed to balance surface accuracy with smoothness and mesh quality, using an edge detection algorithm on labelled voxel data. A different approach is presented in [26], wherein surfaces are meshed away from the junctions, leaving behind void regions which are then meshed using the previously created surface meshes as constraints.
- Finally, when the surfaces are represented via other means, different approaches are available. For example, in [27], point clouds of surfaces are transformed into implicit functions via partitions of unity, before executing a marching tetrahedra variant to extract a mesh. Lastly, in the volume-of-fluid method for tracking interfaces in multiphase fluid flow, the volume fraction of every fluid in each grid cell is tracked, and to evolve the fluids, the interface must be reconstructed from the volume fractions, see e.g. [28].

In comparison, for the mesh-generation algorithm presented here, an initial mesh is constructed using a simple marching tetrahedra-style algorithm that does not require special material-dependent lookup tables. Instead, the Voronoi interface, together with properties of piecewise linear interpolation, guarantees topological consistency of the mesh. In this fashion, heuristic or complex decisions about surface topology are avoided. In the second stage of the algorithm, vertices of the mesh are dynamically moved to improve mesh quality, in part based on the ideas presented in the *DistMesh* algorithm [29]. Here, a specially designed "clamping" function is used to automatically prevent triangle inversion, while allowing low-quality elements to quickly change shape into good-quality elements. The result is a robust and efficient algorithm which can be used in a wide variety of situations.

3 The Voronoi interface

Although the concept of the Voronoi interface can be applied to quite general situations, we shall first motivate it as a type of generalisation of the Voronoi diagram. Given a set of m nodes in $\mathbb{R}^n$, the Voronoi diagram decomposes $\mathbb{R}^n$ into m different cells, such that all the points in a given cell are closer to one particular node than any other. The boundary between these cells is the set of points that are equidistant to at least two nodes, and no closer to any other node. Note that we can obtain a similar method of decomposition if the nodes are replaced by any set of non-overlapping regions Ω_i. In this case, the cells (i.e. "phases") consist of all points that are closer to one particular region than any other; see Fig. 1 for a two-dimensional illustration. As in [2], we define the *Voronoi interface* to be the boundary between these cells/phases, and denote it as Γ_V.

Suppose we can calculate (or approximate) the signed distance function, ϕ_i, to the boundary of each region Ω_i, i.e.

$$\phi_i(x) = \pm \min_{y \in \partial\Omega_i} \|x - y\|,$$

with the sign chosen such that ϕ_i is positive inside Ω_i and negative outside. Using these signed distance functions, the Voronoi interface Γ_V inside a given domain Ω is given by

$$\Gamma_V = \left\{ x \in \Omega : \exists i \neq j \text{ such that } \phi_i(x) = \phi_j(x) \geq \max_{k \neq i,j} \phi_k(x) \right\}. \tag{1}$$

In particular, Γ_V can be separated into a union of individual surfaces separating pairs of different regions, such that $\Gamma_V = \bigcup_{i \neq j} \Gamma_{ij}$, where

$$\Gamma_{ij} = \left\{ x \in \Omega : \phi_i(x) = \phi_j(x) \geq \max_{k \neq i,j} \phi_k(x) \right\}. \tag{2}$$

Although distances were used as motivation, note that (1) and (2) can be used to define an interface, even when the ϕ_i functions are not necessarily signed distance functions. All that is required is that the functions ϕ_i are continuous, and that the individual sets Γ_{ij} are codimension-one surfaces.[1] Therefore, it is useful to make a generalisation: for any set of functions $\phi_i : \mathbb{R}^n \to \mathbb{R}$, the *Voronoi interface of the functions* ϕ_i is defined to be Γ_V given by (1).

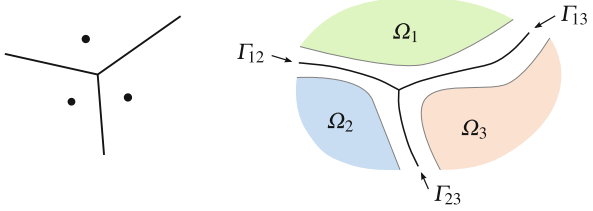

Fig. 1 (Left) Voronoi diagram of three points in the plane. (Right) The Voronoi interface $\Gamma_V = \Gamma_{12} \cup \Gamma_{13} \cup \Gamma_{23}$ corresponding to three given regions Ω_1, Ω_2, and Ω_3. Here, $\mathbb{R}^2$ is divided into three cells (i.e. phases) separated by Γ_V

Defining the interface in this manner, i.e. implicitly rather than explicitly, provides many virtues. For example, in an evolving multiphase system, implicit representations of the interface automatically handle topological changes in the interfaces (such as creation and disappearance of phases, or changes in connectivity) [2, 4]. In relation to the meshing problem, as shown below, this particular implicit representation makes it straightforward to extract a topologically consistent mesh with no artefacts at junctions. Assuming continuity of the functions ϕ_i, an equivalent characterisation of the interface is that a point x is inside phase/material i if and only if $\phi_i(x) > \max_{j \neq i} \phi_j(x)$. This characterisation was used in some previous works on meshing multiple surfaces (wherein the functions were smoothed characteristic/indicator functions); here, (1) is used directly.

In practice, there are many possible methods for defining the functions ϕ_i, such as using closed-form mathematical expressions and/or defining function values on a background grid. In the case that the functions are signed distance functions of implicitly defined geometries, their values can be computed efficiently using a variety of methods, such as the Fast Marching Method [30, 31] (see also the method by Tsitsiklis [32]). In other cases, the functions could be derived from a single label/indicator function $\chi : \Omega \to \mathbb{N}$ that divides the domain into different regions. For example, one could define ϕ_i as a (possibly smoothed) per-phase indicator function such that $\phi_i(x) = 1$ inside region i and $\phi_i(x) = 0$ outside. The framework is flexible, and the particular method of determining ϕ_i depends heavily on the application. In the following, all that is assumed is that the ϕ_i functions are defined on a background grid (such as a regular Cartesian grid). However, it is important to note that it is not necessary to define every ϕ_i function everywhere in the entire domain: in the meshing algorithm, it is only necessary to know the values of ϕ_i in a small narrow band surrounding the boundary of phase i. This can dramatically improve efficiency and is a common technique used, for example, in narrow band level set methods [33].

[1] For simplicity, in the following, interfaces are assumed to contain no holes. However, holes could be made by creating additional phases in the multiphase system whose purposes are to implement void regions, by using additional ϕ_i functions. In a simple implementation, the meshing algorithm may then proceed unaltered and any surface meshes of the void phases may be ignored or discarded upon completion. Alternatively, to improve efficiency, one could make minor modifications to the algorithm such that surface elements of void phases are never created.

4 Mesh generation

Given N functions ϕ_i defined on some three-dimensional grid, the goal is to extract an explicit representation of the Voronoi interface, defined by (1), as a high-quality triangulated mesh. This is accomplished by first creating an initial mesh, which although will generally be of low quality, will nevertheless be topologically consistent in the sense that triangles meet at junctions without overlap or gaps. In the second stage of the algorithm, the mesh is iteratively improved using a sequence of force-based smoothing, projection, and edge-flipping steps. The method is summarised in Algorithm 1, and, in the next set of sections, the implementation of the individual steps is described.

Algorithm 1 General algorithm for mesh generation

```
Create initial mesh.
repeat
    Apply forces and project vertices.
    Edge flip triangles.
until desired mesh quality is achieved.
```

4.1 Creating the initial mesh

The procedure to create an initial mesh of the network of interfaces is based on the approach first presented in [4]. In this method, the Voronoi interface of the multiphase system is extracted using a marching tetrahedra-style algorithm that involves creation and "chopping" of mesh elements. Given the functions ϕ_i, defined on a background grid (such as a regular Cartesian grid), to extract a triangular mesh representation of Γ_{ij} as given by (2), the mathematical procedure consists of three steps:

(i) First, establish a continuous piecewise linear interpolation of every ϕ_i function, to determine ϕ_i at arbitrary points. For example, if the background grid is a regular Cartesian grid, then each cell can be divided into six tetrahedra, as shown in Fig. 2 (left), and in each tetrahedron we can use the obvious linear interpolant of ϕ_i. This is precisely the interpolant that is implicitly used by the marching tetrahedra algorithm. Alternatively, if the background grid is already a tetrahedral mesh, we can directly use the canonical linear interpolant of ϕ_i.

(ii) Next, extract the zero level set of $\phi_i - \phi_j$ as a collection of planar polygon surface elements $\{E_\ell\}_{i=1}^{n}$.[2] Since the last condition in (2) is not necessarily satisfied on every surface element, it follows that $\Gamma_{ij} \subseteq \bigcup_\ell E_\ell$.

(iii) For each element E_ℓ, keep the set of points $x \in E_\ell$ that satisfy $\phi_i(x) = \phi_j(x) \geq \phi_k(x)$ for all $k \neq i, j$. This is achieved by a series of "chop" operations that takes E_ℓ, chops it using the zero level set of $\phi_i - \phi_k$ as the position of the cut, and keeps the piece on which $\phi_i \geq \phi_k$. The piece on which $\phi_i < \phi_k$ is thrown away. Such chopping is made possible by the fact that every level set function ϕ_i is piecewise linear; hence particular level sets as well as intersections of level sets are always linear. In particular, it can be shown that throughout the chopping process, the element is always a convex planar polygon. After chopping is complete, the polygon E_ℓ is then converted into a collection of triangles.

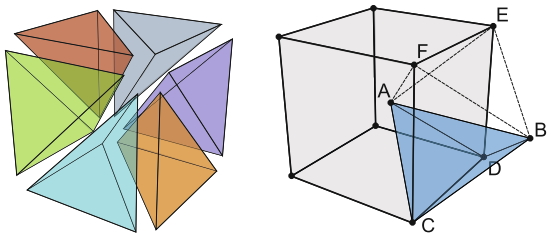

Fig. 2 (*Left*) Dividing a cube into six tetrahedra. (*Right*) A body-centred cubic lattice tessellates space into identical tetrahedra; each tetrahedra has two vertices at the centroid of neighbouring cells, and the other two vertices on an edge of a cell. In the figure, the blue tetrahedron ABCD is one of four sharing a face; the other three are ABCF, ABDE and ABEF

The result of the above procedure is a collection of triangles whose union is Γ_{ij}. Note that the algorithm is *exact*, in the sense that the Voronoi interface of the interpolated multiphase system is extracted exactly. Thus, the topology of the mesh interface coincides precisely with the topology of the Voronoi interface induced by the piecewise linear interpolation of the ϕ_i functions—no ad hoc decisions about mesh topology are made. It follows that different interfaces Γ_{ij} and Γ_{kl} which meet at higher order junctions do so without any overlap or gaps. In addition, triangles that meet at junctions do so by sharing edges and vertices along those junctions. Note also that an arbitrary number of phases/materials can interact within a particular tetrahedron: the Voronoi interface consistently defines the interface throughout the interior of each tetrahedron, and this resolves some of the ambiguity problems in surface topology found in other works.

In the work presented here, the functions ϕ_i are defined on a regular Cartesian grid. If each grid cell is divided into six tetrahedra, see Fig. 2 (*left*), as per a common variant of

[2] Note that $\phi_i - \phi_j$ is a continuous piecewise linear function defined on Ω, and so its zero level set must necessarily be piecewise planar. In particular, because $\phi_i - \phi_j$ is linear on each tetrahedron, it follows that its zero level set in each tetrahedron, if it exists, is either a triangle or a planar quadrilateral.

the marching tetrahedra algorithm, it is almost always the case that the resulting triangulation is extremely poor: many triangles are almost degenerate (small in diameter or slivers) and there is often many more triangles than necessary to capture the features of the interface, as demonstrated in Fig. 3 (*top*). To improve this, two complementary ideas are used:

- Instead of dividing each grid cell into six tetrahedra, we have used a "body-centred cubic (BCC)" lattice, as shown in Fig. 2 (right). In particular, we have used a BCC lattice which requires no interpolation by skipping every second grid point of the background reference Cartesian grid: in Fig. 2 (right), grid points precisely halfway between C and D, D and E, E and F, F and C, etc., which exist on the background Cartesian grid, are not members of the tessellation. The BCC lattice has been widely used in the literature (see for instance [8, 19, 20, 25]) as the tetrahedra have identical shapes and sizes, are more evenly distributed, and lead to fewer grid-dependent effects, as compared to the 6-tetrahedron split in the standard marching tetrahedra algorithm. These effects, as well as other possible subdivisions of Cartesian grids into tetrahedra, are extensively analysed in [34].
- In addition, we have used a vertex snapping procedure which eliminates sliver triangles. For a single scalar function ϕ defined on the background grid's vertices, assuming the zero level set is sought, vertex snapping slightly perturbs ϕ by setting the value of ϕ to be zero at any vertices where it is approximately zero:

$$\tilde{\phi(x)} = \begin{cases} 0 & \text{if } |\phi(x)| < \epsilon, \\ \phi(x) & \text{otherwise.} \end{cases}$$

- This effectively snaps mesh vertices that are approximately near the background grid points to be precisely on those grid points, without introducing any holes or artefacts in the mesh. In practice, even for considerable reduction of sliver elements, the threshold ϵ can be very small. In our case, the mesh created by this algorithm is used as the input to an iterative procedure that projects vertices on surfaces and junctions to their constraint manifolds, as explained in the next section. Thus, the initial mesh does not need to be entirely accurate and as a result large tolerances can be used, leading to a significant reduction of unnecessary triangles in the mesh, while maintaining the same feature resolution. When the scalar functions are approximate distance functions, it was found that setting $\epsilon \approx 0.2h$, where h is the Cartesian grid cell size, gave good results. For the multiphase system, vertices can be snapped by perturbing the level set function values in a pairwise fashion:

Algorithm 2 Creating the initial Voronoi interface mesh

1: Snap vertex values.
2: **for** each tetrahedron $\mathscr{T}$ **do**
3: **for** each subset $\{i, j\} \subseteq \{1, \ldots, N\}$ **do**
4: Extract the zero level set of $\phi_i - \phi_j$ inside $\mathscr{T}$ using linear interpolation.
5: Define P to be the resulting convex planar polygon; (if it does not exist, **continue** to next $\{i, j\}$ subset).
6: **for** each $k \neq i, j$ **do**
7: Evaluate $\psi := \phi_i - \phi_k$ at each vertex of P.
8: Cut the polygon at the zero level set of ψ and keep the part on which $\psi \geq 0$.
9: **if** P is now empty, **continue** to next $\{i, j\}$ subset.
10: Divide P into triangles and add to the collection Γ_{ij}.
11: If necessary, uniquify vertices.

for each background grid point x **do**
Define $\tilde{\phi}_i(x) := \phi_i(x)$ for all i.
for $i = 1, \ldots, N$ **do**
for $j = i+1, \ldots, N$ **do**
if $|\tilde{\phi}_i(x) - \tilde{\phi}_j(x)| < \varepsilon$ **then**
$\tilde{\phi}_j(x) \leftarrow \tilde{\phi}_i(x)$

Snapping in a pairwise fashion like this ensures that previous alterations do not get overridden by later alterations, which in turn means the topology of the multiphase interface is not affected (provided it was already sufficiently resolved).

Combining these ideas, we are led to Algorithm 2 for creating an approximation of the Voronoi interface. On line 3, it is only necessary to loop over the pairs of phases defined in the particular tetrahedron being considered. On lines 4 and 5, a lookup table, similar to those used in a standard marching tetrahedra algorithm, can be used to determine how to extract the polygon (as either a triangle or quadrilateral). On line 7, a simple method to evaluate ψ at arbitrary locations is to use pre-computed Lagrange basis functions, while on line 8, it is a simple exercise to design an algorithm to cut a convex planar polygon by the zero level set of a linear function defined on its vertices. Polygons are then dissected into triangles and added to the overall collection of triangles. These steps are straightforward if one is not concerned about duplicating vertices, i.e. triangles are represented as 3-tuples of vectors in $\mathbb{R}^3$. Clearly, this is not optimal, since representing a triangle with a 3-tuple of vertex indices is more efficient. So, on line 11, the collection of vertices are uniquified, by identifying vertices as equal if they are within a small amount of machine precision

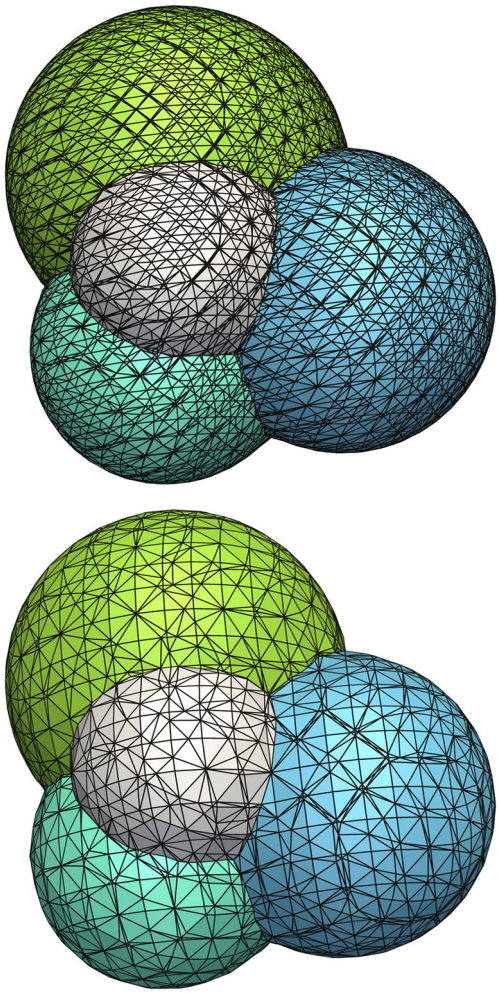

Fig. 3 A multiphase mesh of a five-phase system of four spheres, using the polygon chopping method (Algorithm 2). Four pairwise surfaces and three quadruple point junctions are visible. (*Top*) Using the standard decomposition of dividing a grid cell into six tetrahedra, without vertex snapping. (*Bottom*) Using the body-centred cubic lattice, with vertex snapping

round off error.[3] Alternatively, one may uniquify vertices simultaneously with building the collection of triangles.

The result of the mesh creation algorithm on an example multiphase system is shown in Fig. 3. Here, a five-phase system is defined by setting $\{\phi_i\}_{i=1}^4$ to be signed distance functions for four differently positioned spheres, i.e. $\phi_i(x) = r_i - \|x - x_i\|$, and then defining

$$\phi_5 = \min(-\phi_1, -\phi_2, -\phi_3, -\phi_4)$$

to be the exterior phase. This system has a total of five phases, ten individual surfaces Γ_{ij} separating pairs of phases, ten distinct triple line junctions, and five quadruple point junctions. (Only some of these surfaces/junctions are visible in the figures; see also Fig. 6 for a partial cutaway view.) Figure 3 (top) shows the mesh obtained by using the standard decomposition of each Cartesian grid cell into six tetrahedra, without vertex snapping. In comparison, Fig. 3 (bottom) shows the result using the body-centred cubic lattice, with vertex snapping. We can see that there is a significant reduction in sliver triangles, as well as the overall triangle count.

4.2 Smoothing and projection

Once an initial mesh of the Voronoi interface of the multiphase system is created, a high-quality mesh can be obtained with mesh quality improvement techniques. These techniques have been extensively studied and come in a variety of forms, involving a combination of vertex movements, updates in mesh topology via edge flipping, and refinement techniques. For example, in [35], volumetric tetrahedral meshes of multi-material domains are improved by classifying mesh vertices as fixed, on triple lines, on surfaces, or in the interior, and different strategies are applied to each, ranging from average mean curvature flow to gradient descent on an energy functional measuring quality factors. These techniques can be made efficient, at the price of a more involved implementation, by employing a backward Euler strategy and using implicit techniques [36]. In this work, motivated by simplicity of implementation, we adopt an explicit strategy that uses force-based smoothing, projection of vertices onto their constraining surfaces, and edge flipping, made efficient with a locally adaptive time-stepping procedure. In particular, the ideas underlying the DistMesh algorithm [29] are extended and adapted to the case of interconnected surfaces. Vertices of the mesh are moved according to "forces" exerted on them by the edges in the mesh; in the original DistMesh algorithm, the edges are analogous to springs that resist compression when shorter than a certain rest-length, but do not otherwise resist expansion. It was observed [29, 37] that this repulsive force combines exceptionally well with edge flipping (or regular Delaunay triangulation) to quickly generate meshes with very good connectivity properties.

Since this process predominantly involves calculations involving vertices and their edges, it is advantageous to use a vertex-and-edges data structure. Hence, let $x_i \in \mathbb{R}^3$, $i = 1, \ldots, n$, denote the mesh vertices and $\mathscr{E}_i \subset \mathbb{N}$ denote the set of neighbours of vertex i, so that (x_i, x_j) is an edge for each $j \in \mathscr{E}_i$. From the creation of the initial mesh, we also know which surfaces each vertex is situated on. For each vertex x_i, let $\chi_i \subset \mathbb{N}$ denote the set of phases for

[3] With mild assumptions on the functions ϕ_i, it is possible to show that the vertex snapping procedure guarantees that if two vertices on a surface are within a distance ϵ from each other, such that $\epsilon \ll h$ (where h is the tetrahedron length), then they are in fact the *same* vertex. In this work, a tolerance of $\epsilon = 10^{-14}$ was used (corresponding to double precision and unit-length domains). Extensive tests analysing separation distance of vertices found that this tolerance correctly uniquified vertices in all cases.

which x_i is on the boundary. Thus, if χ_i has two, three, or four elements, then x_i is on a surface, a triple junction, or a quadruple point, respectively. At times, it is also necessary to determine the set of triangles connected to a particular vertex; denote this set as $\mathscr{T}_i$. Determining $\mathscr{T}_i$ can be done on-the-fly by manipulating the edge data structure.

In the force-based smoothing strategy, each edge (i, j) exerts a force $f_{ij} = -f_{ji}$ on vertex i. The goal is to find an equilibrium such that at each vertex, the net force is zero. Solving such a nonlinear system is a relatively difficult task, so instead the vertices are moved iteratively, using a simple forward Euler analogy:

$$x_i^{n+1} = x_i^n + \Delta t \sum_{j \in \mathscr{E}_i} f_{ij}.$$

Here, Δt is an artificial time step that controls the progress of the mesh towards the final desired state. Note, however, that if Δt is too large, mesh elements can invert and this can cause the iterative process to become wildly unstable. On the other hand, if Δt is too small, then efficiency can be sacrificed since too little progress may be made. A solution to this problem is presented later, using a method that essentially locally computes the time step according to a basic stability condition that prevents inversion of triangles, and leads to rapid convergence.

4.2.1 Force functions

In [29], the force function represented edges as springs that resist compression when shorter than a certain rest-length, and do not resist expansion when longer. Despite being a simple approach, it was shown that such a force quickly leads to a mesh with uniformly high-quality elements. For an edge (x, y), the force exerted on x is defined by

$$f_D(x, y, \ell_0) = \frac{x - y}{|x - y|} \max(\ell_0 - |x - y|, 0),$$

where ℓ_0 is a rest-length that is related to the desired average edge length of the final mesh. In fact, it is possible to allow ℓ_0 to vary spatially, allowing the mesh to automatically refine where necessary. However, in this work, a mesh which has uniform triangle sizes throughout is sought, and so ℓ_0 will be spatially uniform. As noted in [29], it is important for vertices of the mesh to spread out across the whole geometry, which means that the rest-length ℓ_0 should be slightly larger than the actual desired edge length in the mesh. This is achieved with a simple scaling: at the beginning of each iteration, the average edge length (measured in a L^2 norm) is computed and scaled by a factor[4] of 1.2 to define

$$\ell_0 := 1.2 \left(\frac{\sum_{i=1}^n \sum_{j \in \mathscr{E}_i} \|x_i - x_j\|^2}{\sum_{i=1}^n |\mathscr{E}_i|} \right)^{1/2}. \quad (3)$$

This repulsive force ties in well with edge flipping and quickly leads to high-quality elements. However, when edge flipping is not possible, as is the case on triple junctions, the repulsive force can lead to situations in which vertices attempt to exit their constraining surfaces, causing mesh elements to invert. Instead, it was observed that Laplacian smoothing works very well on, and near, junctions. In this case, an attractive force f_L is used, where

$$f_L(x, y) = \frac{1}{2}(y - x).$$

Correctly normalised by the number of edges, the attractive force is equivalent to a half-step of Laplacian smoothing, which acts to move vertices to the average location of its neighbours. This force is used for all junction vertices, as well as surface vertices that have at least one neighbour on a junction.

4.2.2 Projection

A side effect of the force-based smoothing is that over time, mesh vertices can deviate from the surfaces on which they are meant to be constrained, i.e. the mesh vertices can stray from the Voronoi interface. To fix this, one can *project* the vertices back onto the surfaces on which they belong, by moving them in a direction orthogonal to the constraining surface. Such a scheme was used in [29, 37] for vertices constrained to live on codimension-one surfaces, and a different scheme was used in [18] for triple junctions. Note that this procedure also fixes the small aberrations caused by the vertex snapping used in creating the initial mesh.[5] For a single function, a vertex x close to the zero level set of ϕ is (approximately) projected onto the zero level set with the update

$$x \leftarrow x - \frac{\phi(x)\nabla\phi(x)}{|\nabla\phi(x)|^2}.$$

The update can be viewed as moving x to its closest point on the zero level set of the linear approximation of ϕ at x, given by $\phi(x + \delta) \approx \phi(x) + \delta \cdot \nabla\phi(x)$. In the case of a vertex belonging to a triple junction, or a higher order constraint set, one would like to project x to the corresponding multi-junction interface. To do this, the method for a single level set function can be generalised to multiple level set functions by considering pairwise

[4] The scaling factor of 1.2 was determined empirically, and is the same as that used in [29]. It forces mesh vertices to spread apart across the whole surface and leads to a smoothing behaviour that performs consistently well.

[5] In the projection step, the original, unperturbed functions ϕ_i are used, i.e. they have not been altered by the vertex snapping procedure used in creating the initial mesh.

combinations: for a vertex x belonging to the boundary of phases $\chi \subseteq \{1\ldots,N\}$, define

$$p(x,\chi) = -\frac{1}{|\chi|-1}\sum_{i\in\chi}\sum_{j\in\chi,\,j>i}\frac{\big(\phi_i(x)-\phi_j(x)\big)\big(\nabla\phi_i(x)-\nabla\phi_j(x)\big)}{|\nabla\phi_i(x)-\nabla\phi_j(x)|^2}. \quad (4)$$

In essence, the function accumulates the shifts arising from projection onto individual surfaces Γ_{ij}, such that one iteration is given by $x \leftarrow x + p(x,\chi)$. Note that (4) reduces to the case of a single surface when $|\chi| = 2$, and also that $p(x,\chi) = 0$ whenever x is already on the multi-junction interface (i.e. $\phi_i(x) = \phi_j(x)$ for all $i,j \in \chi$). The normalisation factor in (4) is designed to give efficient convergence without causing oscillations (i.e. by ensuring the steps are not too large). Finally, we mention that special treatment may be needed if any of the denominators in (4) are close to zero; this never caused problems in any of our tests, but may happen if the functions ϕ_i are very poorly defined. In such cases, possible methods to fix this could include reinitialising ϕ_i as signed distance functions for each phase.

Note that the gradient of the functions ϕ_i (as well as the function values themselves) must be somehow calculated or approximated. If they are given by closed-form mathematical expressions, then one could calculate the gradients using these expressions. Alternatively, if the functions are defined on a background Cartesian grid, then one could use, for example, standard second-order finite differences to calculate their gradients at grid points, followed by trilinear interpolation to evaluate the gradients and function values anywhere within the domain. This latter method was adopted in much of the work presented here.

4.2.3 Adaptive time stepping

The net result of the force-based smoothing and projection leads to displacements of mesh vertices in the form

$$x_i \leftarrow x_i + \delta_i.$$

Here, δ_i is a displacement vector that is the sum of the forces, together with the projection. If this displacement is too large, mesh elements can invert, causing the smoothing process to become unstable. To solve this, a simple clamping algorithm can be used, wherein δ_i is made sufficiently small in such a way that no triangle inverts. The algorithm is designed to allow vertex x_i to move "as far as it can" in the direction δ_i, so that a small edge or triangle can quickly expand if the forces want it to. The clamping is performed for each vertex independently of all other vertices, and effectively establishes an easy-to-implement time stepping that is locally adaptive. Specifically, for a particular vertex i, each of the triangles connected to i are used to clamp by the necessary amount, to form the replacement

$$\delta_i \leftarrow \delta_i \times \min\Big(1, \min_{t\in\mathcal{T}_i}\frac{1}{2}\mathit{clamp}(\delta_i, x_i, t)\Big). \quad (5)$$

Here, *clamp* is a function that returns the maximum amount by which x_i can be moved in the direction δ_i without inverting the triangle t, independently of how the other vertices of the triangle move. The function is implemented as follows. Consider Fig. 4, which shows a triangle having vertices v_0, v_1, v_2. The indicated lines are constructed by connecting midpoints of the three edges. The lines divide the plane into three disjoint regions that are separated by the shaded region indicated in Fig. 4. If it can be guaran-

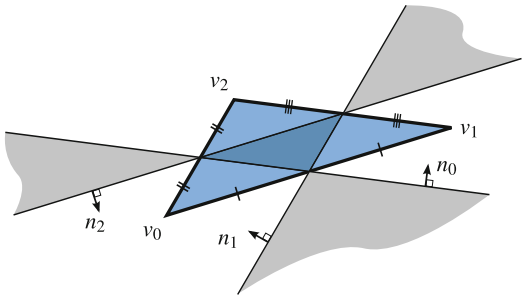

Fig. 4 Lines passing through the midpoints of a triangle (with vertices v_0, v_1, v_2) divide the plane into the shaded region (having four separate components) and non-shaded region (having three separate components), such that the line opposite vertex v_i has normal n_i. In the clamping procedure, each of the vertices v_i is allowed to move anywhere inside its own particular component of the non-shaded region

Algorithm 3 Calculating $\mathsf{clamp}(\delta, v_0, t)$ where t is the triangle with vertices v_0, v_1, v_2

Compute the three normal directions

$$n_0 = v_2 - v_0 - \tfrac{(v_2-v_0)\cdot(v_2-v_1)}{\|v_2-v_1\|^2}(v_2-v_1),$$
$$n_1 = v_0 - v_1 - \tfrac{(v_0-v_1)\cdot(v_0-v_2)}{\|v_0-v_2\|^2}(v_0-v_2),$$
$$n_2 = v_1 - v_2 - \tfrac{(v_1-v_2)\cdot(v_1-v_0)}{\|v_1-v_0\|^2}(v_1-v_0).$$

return s calculated by

$$s = \min\Big(\frac{n_0\cdot(\frac{1}{2}(v_0+v_1)-v_0)}{\max(n_0\cdot\delta, 0^+)}, \frac{n_1\cdot(\frac{1}{2}(v_0+v_1)-v_0)}{\max(n_1\cdot\delta, 0^+)}, \frac{n_2\cdot(\frac{1}{2}(v_2+v_0)-v_0)}{\max(n_2\cdot\delta, 0^+)}\Big),$$

where it is understood that if one of the max statements evaluates to 0^+, then the corresponding fraction is $+\infty$.

Algorithm 4 A single smooth and projection step

1: Calculate the rest length ℓ_0 using (3).
2: **for** $i = 1, \ldots, n$ **do**
3: Calculate the force on vertex i due to its neighbours $\mathscr{E}_i$:

$$\delta_i = \begin{cases} 0 & \text{if vertex } i \text{ is a high-order junction } (|\chi_i| > 3), \\ \left(\sum\limits_{\substack{j \in \mathscr{E}_j \\ \chi_i \subseteq \chi_j}} 1 \right)^{-1} \sum\limits_{\substack{j \in \mathscr{E}_j \\ \chi_i \subseteq \chi_j}} f_L(x_i, x_j) & \text{else if vertex } i \text{ is on a triple junction } (|\chi_i| = 3), \\ & \text{or has a neighbour on a triple junction,} \\ \dfrac{1}{|\mathscr{E}_j|} \sum\limits_{j \in \mathscr{E}_j} f_D(x_i, x_j) & \text{otherwise.} \end{cases}$$

4: **if** vertex i is on a surface ($|\chi_i| = 2$) **then**
5: Remove the component of δ_i orthogonal to the surface: $\delta_i \leftarrow \delta_i - (\delta_i \cdot n_{\chi_i}) n_{\chi_i}$.
6: Add to δ_i the projection displacement: $\delta_i \leftarrow \delta_i + p(x_i, \chi_i)$ using (4).
7: Clamp δ_i to prevent inversion using (5) and Algorithm 3.
8: **for** $i = 1, \ldots, n$ **do**
9: $x_i \leftarrow x_i + \delta_i$

teed that the vertices v_0, v_1, v_2 move only in their respective regions, without crossing the shaded region, then the triangle will not invert. In full three-dimensional space, these constraint regions are convex, bounded by the planes with the indicated normal vectors, which in turn are parallel to the plane of the triangle.

Consider then a particular vertex, v_0, say, with a corresponding desired displacement vector δ. The minimum $s > 0$ (if it exists) is computed such that $v_0 + s\delta$ is on the boundary of its constraint region. For a particular plane with normal n containing the point a, it follows that $(x + s\delta - a) \cdot n = 0$, giving $s = (a - x) \cdot n / \delta \cdot n$. By doing this for each appropriate plane, this leads to Algorithm 3 to compute the clamp function for a particular vertex v_0 of a specified triangle, with the specified displacement δ.

Note that in (5), the clamping is reduced by a factor of two, thereby preventing triangle inversion entirely. As a result, triangles may become nearly degenerate only after an accumulation of displacements over several iterations. In this scenario, the clamping factor and local "time step" would reduce to zero and convergence would halt. However, this event never occurs in practice, precisely because edge flipping changes the topology of the mesh so that any nearly degenerate triangles are removed.

4.2.4 Combining smoothing and projection

Putting all of the above together, Algorithm 4 performs one iteration of force-based smoothing and projection, with the adaptive time stepping. On line 3, in the second item of the case statement, only the neighbours which belong to the same type of surface (via the conditional $\chi_i \subseteq \chi_j$) are considered. This ensures that the force law involves only edges belonging to the same type of surface (or triple junction). On line 5, the force computed from the edges is enforced to be tangential to the surface, by removing the component of the vector orthogonal to the surface. (Here, n_{χ_i} is a normal vector to the surface $\chi_i = \{j, k\}$, i.e. is proportional to $\nabla\phi_j - \nabla\phi_k$, and can be calculated with finite differences.) Without this, it was observed that the non-tangential components of the force can start to compete with projection, causing oscillation. Hence, the force-based smoothing is restricted to tangential forces.

4.3 Edge flipping

In the edge-flipping step, every pair of triangles sharing a common edge is considered. If flipping the shared edge (see Fig. 5) improves the quality (defined below) of the pair of triangles, then this is done and the connectivity of the mesh is updated, before visiting other edges. In the plane, and with a quality measure that is based on the standard Delaunay in-circle condition, this iterative procedure is a well-known method that converges to a Delaunay triangulation [38]. A simple measurement of triangle quality that is suitable for triangles in $\mathbb{R}^3$ is to define

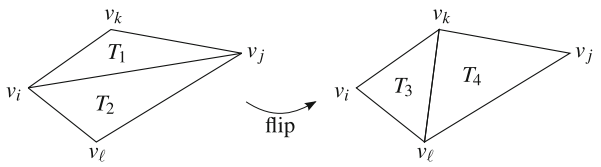

Fig. 5 Edge-flipping exchanges an edge shared by two triangles T_1 and T_2 (*left*) with the edge formed by opposing vertices, giving two new triangles T_3 and T_4 (*right*)

Algorithm 5 One iteration of edge flipping

1: **for** each edge (i, j) in the mesh **do**
2: **if** the edge is shared by two triangles that belong to the same surface **then**
3: Let v_i, v_j, v_k and v_i, v_ℓ, v_j be the vertices of the two triangles (see Figure 5).
4: **else**
5: **continue** to next edge.
6: Compute $q_1 = q(v_i, v_j, v_k)$ and $q_2 = q(v_i, v_\ell, v_j)$.
7: **if** $\min(q_1, q_2) \geq 0.9$ **then**
8: **continue** to next edge *(since the triangles are already of good quality)*.
9: Compute $q_3 = q(v_i, v_\ell, v_k)$ and $q_4 = q(v_\ell, v_j, v_k)$.
10: **if** $\min(q_3, q_4) \leq \min(q_1, q_2)$ **then**
11: **continue** to next edge *(since edge flipping will not improve quality)*.
12: Let n_3 and n_4 be proportional to the normal of the two new triangles:
$$n_3 = (v_\ell - v_i) \times (v_k - v_i),$$
$$n_4 = (v_k - v_j) \times (v_\ell - v_j).$$
13: **if** $n_3 \cdot n_4 \leq 0$ **then**
14: **continue** to next edge *(since edge flipping will invert the triangles)*.
15: Proceed with edge flip: remove edge (i, j) and add edge (k, ℓ) by altering $\mathscr{E}_i, \mathscr{E}_j, \mathscr{E}_k$, and $\mathscr{E}_\ell$.

$$q = q(v_0, v_1, v_2) = 4\sqrt{3}\frac{\text{triangle area}}{\text{sum of squared edge lengths}} = 2\sqrt{3}\frac{\|(v_1 - v_0) \times (v_2 - v_0)\|}{\|v_1 - v_0\|^2 + \|v_2 - v_1\|^2 + \|v_0 - v_2\|^2},$$

where v_i are the vertices of the triangle. Here, q is normalised so that $0 \leq q \leq 1$: an equilateral triangle has $q = 1$ and a degenerate triangle has $q = 0$. (There are also many other types of measures of element quality, see the review [39].) Even though the triangles generally lie on a curved surface, the edge-flipping algorithm is essentially equivalent to the logic used in the plane; the algorithm is summarised in Algorithm 5. Note that on line 2 of Algorithm 5, the potential flip is required to only involve triangles belonging to the same surface. This ensures that edges on mesh junctions do not change connectivity. In addition, line 7 checks to see if the triangle pair is already of good quality; if they are, then any possible edge flipping is essentially inconsequential, since any increases in quality would be minor. This simple check leads to markedly better efficiency in the algorithm.

4.4 Parallelisation

Combining the above steps, i.e. Algorithm 1, yields the basic algorithm to generate a triangular mesh of the Voronoi interface. In some applications, the Voronoi interface is determined by the evolution of complex physics on high resolution three-dimensional grids. For example, in [1], the liquid and gas dynamics in a soap bubble foam were modelled, requiring computation on hundreds of processors. In particular, the computational domain was divided into subdomains, which were then assigned to individual processors in an MPI implementation. In such cases, the functions which define the Voronoi interface are split across several subdomains, and it follows that the mesh-generation algorithm must also be parallelised to maintain some level of computational efficiency. This can be done with a few modifications to the individual steps of the algorithm, mainly in relation to synchronisation of mesh connectivity information across processor boundaries, as follows.

In the first step, an initial mesh approximating the Voronoi interface is found. This step is essentially unchanged: each processor can create a set of triangles from the functions defined on its subdomain. It is only necessary to ensure elements are not duplicated across boundaries shared between subdomains. In a synchronisation step, the processors then communicate and assign unique identifiers to the mesh vertices found in the first step.

At this stage, only mesh smoothing and edge flipping remain, with many possible methods of parallelisation. In this work, the implementation has been simplified by keeping the same domain decomposition and processor layout. In other words, each vertex of the mesh is assigned ownership to a particular processor, according to the same subdomain decomposition. With this simple approach, the smoothing and edge flipping can be parallelised by

1. re-assigning ownership of vertices whenever they move across subdomain boundaries; and
2. maintaining a "ghost layer" of mesh connectivity information, to allow each processor to perform edge-based calculations. That is, each processor maintains a data structure for the vertices it owns, plus any neighbours of those vertices.

Together, these allow each processor to perform one step of force-based smoothing and projection, after which a synchronisation step is needed to update information near the subdomain boundaries.

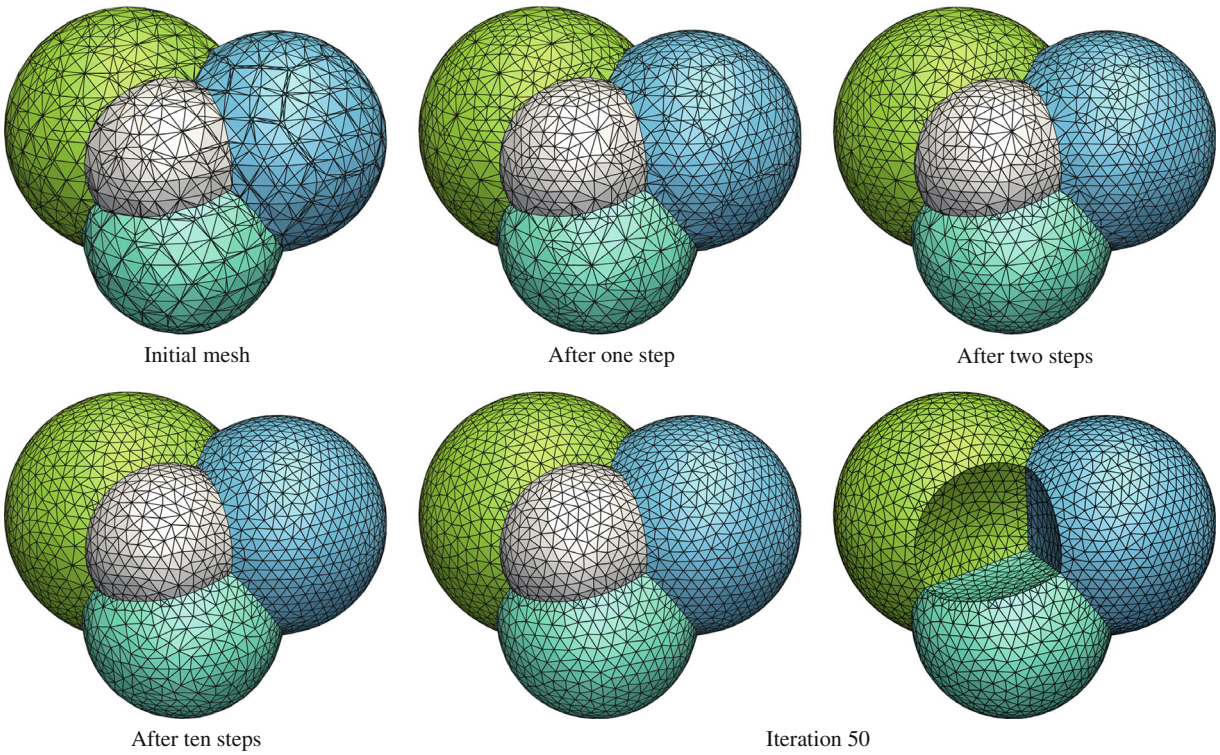

Fig. 6 Using the same initial mesh as in Fig. 3 (*bottom*), the mesh is shown after 1, 2, 10, and 50 steps of smoothing and edge flipping. The *bottom-right* two frames correspond to the same mesh, except in one the middle sphere has been cut away to reveal the interior mesh

The only subtle difficulty with this approach is the need to edge flip across subdomain boundaries. This can be achieved by using two ghost layers, i.e. each processor knows the connectivity and position of each neighbouring vertex of each neighbouring vertex of each vertex the processor owns. Processors then mutually agree before the edge-flipping step, as to exactly which processor performs the edge flipping on triangles spanning a subdomain boundary. By taking this in turn, one can ensure that every pair of triangles in the overall mesh will be subject to edge flipping.

Note however that this simple domain decomposition approach may not necessarily scale with the number of processors. It is entirely possible that the Voronoi interface exists in some parts of the domain, and not in others, so that some processors are assigned no mesh vertices, while other processors are assigned many. It follows that the scaling of this approach depends highly on the geometry of the interface. Therefore, different parallelisation methods may be necessary if scaling is crucial to performance; see, for instance, the general framework developed in [40] for designing scalable mesh improvement algorithms. We remark that in the case of the soap bubble foam application mentioned previously—in which tens of thousands of meshes were generated in the course of one simulation, with each mesh having hundreds of thousands to millions of elements—this simple approach was found to scale very well and was sufficient to make the computational cost of mesh generation only a minute fraction of the overall simulation cost.

5 Results

Consider the example shown in Fig. 3 of four intersecting spheres. Using the same initial mesh [i.e. Fig. 3 (bottom)], Fig. 6 shows successive iterations of smoothing and edge flipping. We see that after just one or two steps, many of the poorly shaped triangles have significantly improved in quality. After ten steps, much of the mesh has been "cleaned up". At this point, the majority of changes in the mesh after more smoothing steps is in edge connectivity alone, rather than in improving element quality. Convergence to an equilibrium is essentially attained after 50 iterations. These observations can be made more quantitative by examining various measures of mesh quality and geometry as a function of iteration count, as shown in Fig. 7. Here, histograms as a function of iteration count are shown, for triangle inradius, circumradius, edge length, quality, angles, and vertex degree. The plots show that poor-quality elements are quickly replaced, and that the distribution of inradii, circumradii, edge lengths, and

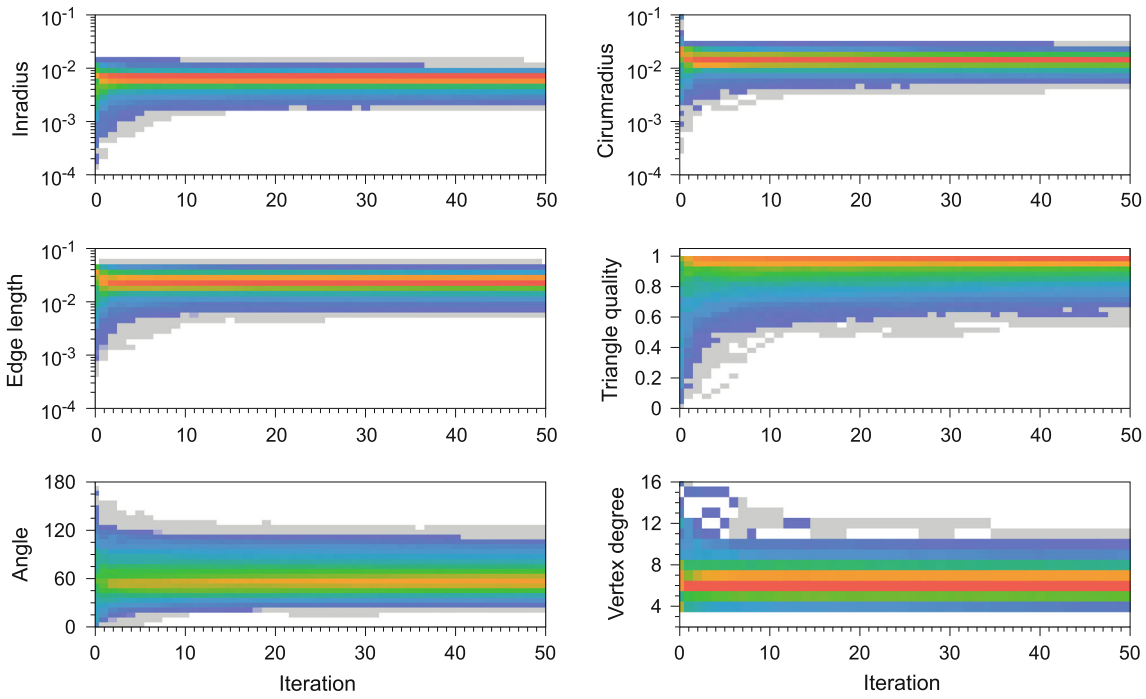

Fig. 7 Measures of mesh quality and geometry as a function of the number of iterations, corresponding to the example shown in Fig. 6. After a specific number of iterations, the corresponding vertical slice in each *graph* is a histogram indicating the percentage of features with the associated measurement: indicates less than 0.1%, indicates 0.1%, 1%, 10%, 20%, and indicates at least 50%

triangle quality converge favourably so that the vast majority of elements have approximately the same geometry. The exception is in the distribution of triangle angles, which has higher variance. In addition, we see that after 20–30 iterations, the measures of mesh geometry have approximately reached an equilibrium. Essentially identical behaviour was seen in every other problem the mesh generator was tested on.

In the next example, the Voronoi Implicit Interface Method [2, 4] has been used to evolve seven randomly created phases under the action of multiphase curvature flow using periodic boundary conditions. Figure 8 shows the result of the meshing algorithm applied to the multiphase system at a particular instant in time. One can see that the meshing algorithm is able to handle complicated geometry, especially so in this case, due to the proximity of some junctions to other junctions, as shown in the magnified portion.

This particular example of a multiphase interface is used to demonstrate the efficiency of the algorithm. A parallelised implementation of the algorithm was used, on a mainstream desktop computer using eight cores (by dividing the cube into $2 \times 2 \times 2$ subdomains). Creating the initial mesh (as described in Sect. 4.1) took 10ms of time. Vertices of the initial mesh were then given unique identifiers across all eight processors, and this synchronisation step took 0.25s, although our implementation of this step could be made faster. For the example shown in Fig. 8, there were 16,592 triangles in total, and 50 iterations of smoothing were used, taking about 1.5s in total. Thus, each iteration took approximately 30 ms time, and it was observed that the individual components contributed: 35% for force-based smoothing and projection; 20% for edge flipping; and 40% for synchronisation among processors. In general, the cost of the mesh-generation algorithm has two components: in the first stage of creating the initial mesh, the cost is linear in the number of tetrahedra visited; in the second stage, each smoothing iteration has a cost linear in the number of mesh vertices.

We conclude this section by demonstrating three more applications of the meshing algorithm.

Figure 9 shows a case that involves defining objects as intersections or unions of others, in a fashion that capitalises on the implicit representation of an interface. A sphere is divided into two halves, with the dividing surface defined implicitly via the zero level set of

$$f(x, y, z) = \cos x \sin y + \cos y \sin z + \cos z \sin x. \quad (6)$$

The three functions are defined as

$$\phi_0(x, y, z) = \sqrt{x^2 + y^2 + z^2} - r + |f(x, y, z)|^3,$$
$$\phi_1(x, y, z) = f(x, y, z)^3,$$
$$\phi_2(x, y, z) = -f(x, y, z)^3,$$

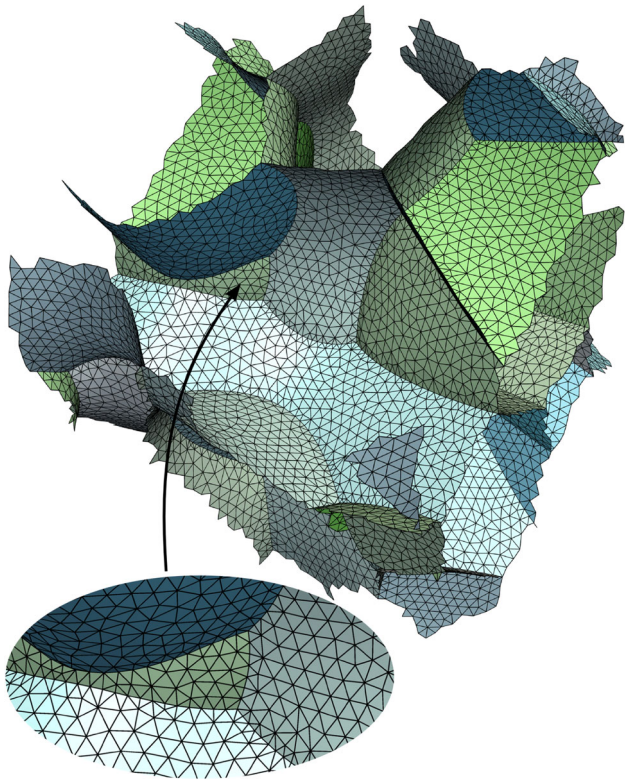

Fig. 8 An instant in time of a multiphase system undergoing curvature flow with volume conservation. The jagged boundary is due only to the periodic boundary conditions

where $r = 3\pi/2$ is the radius of the sphere in this example. As shown in Fig. 9, the algorithm is able to smoothly divide the sphere into two halves, correctly reproducing sharp features.

In the next application, the Voronoi interface is used to reconstruct surfaces from volumetric point cloud data, i.e. a set of scattered points in 3D, such that the points are labelled according to which region they occupy. Point cloud data may arise in various applications, such as in experiments that use tracer particles to study fluid flow patterns, Lagrangian-based multiphase fluid flow simulations, or imaging devices that probe the interior of an object. In this particular example, a cloud of randomly generated points has been created, as shown in Fig. 10 (left). Individual particles have been assigned to one of four regions: red, green, blue, or grey. Let $y_i, i = 1, \ldots, m$, denote the position of the particles and let χ_i denote what region they are in. For each region, define the function ϕ_i which measures the minimum distance to the cloud of region i, i.e.

$$\phi_i(x) := \min_{1 \le j \le m,\, \chi_j = 1} \|x - x_j\|. \tag{7}$$

With this definition, it follows that the Voronoi interface of the functions ϕ_{red}, ϕ_{green}, etc. separates one region from another by a surface going through the middle of the space between them. By computing ϕ_i on a background Cartesian grid, the mesh-generation algorithm can be used to automatically reconstruct these surfaces. For a background grid of size $32 \times 32 \times 32$, Figure 10 shows the reconstructed surfaces, showing nontrivial geometry at the junctions. This type of surface reconstruction from volumetric point cloud data could be especially useful when there are a large number of points since very efficient algorithms can be used to evaluate the functions ϕ_i.

Finally, in Fig. 11, an example is taken from a multiscale model of the gas and liquid dynamics in a foam of soap bubbles [1]. The model can be used to study the collapse of a foam due to membrane rupture, and couples the macroscopic effects of gas dynamics to the microscopic effects of thin-film liquid drainage. In the model, the fluid dynamics of the soapy solution inside the membranes is determined by thin-film equations defined on the network of curved surfaces. This leads to a system of fourth-order nonlinear parabolic PDEs which are coupled at the junctions through nonlinear flux boundary conditions, and in [1], a finite element method was used to solve this system numerically. Since the rearrangement of bubbles is often complex, it becomes necessary to rely on a robust mesh-

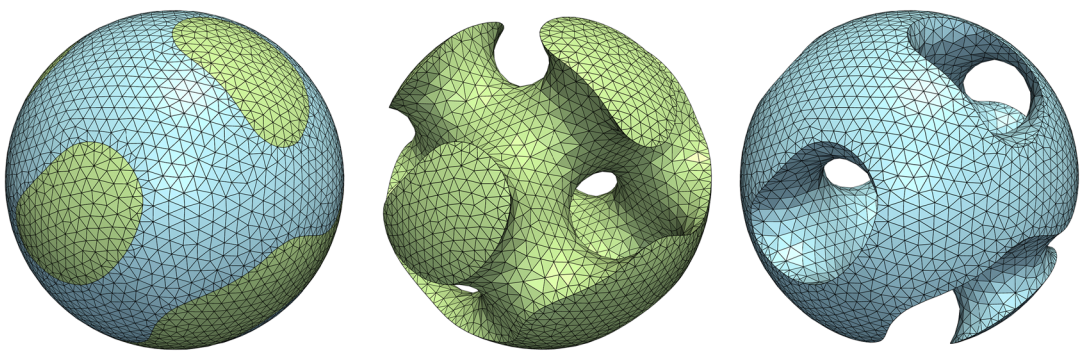

Fig. 9 Dividing a sphere into two parts, with the dividing surface given implicitly by the zero level set of (6). The left figure shows the sphere as a whole, and the right two figures show the individual halves

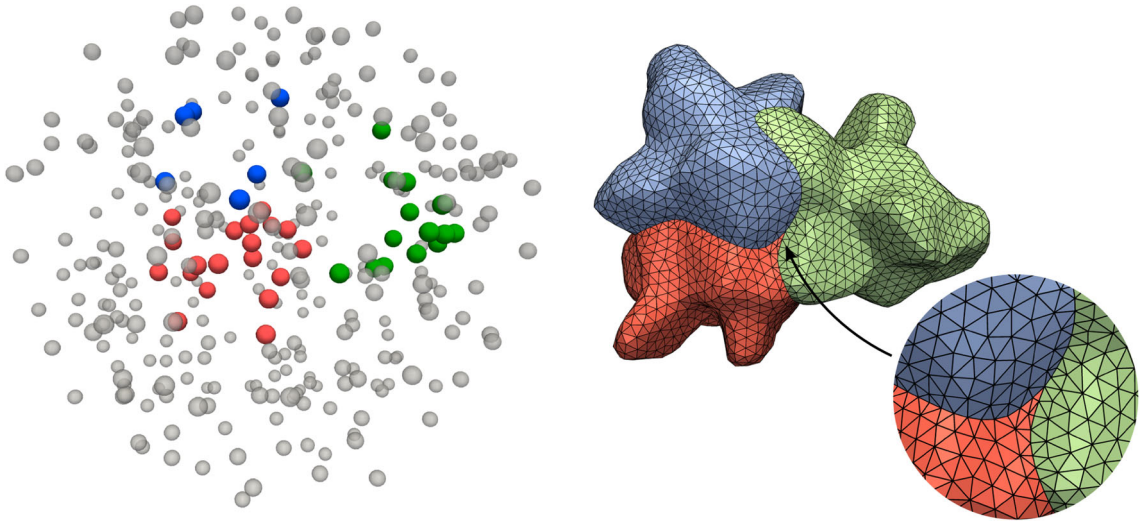

Fig. 10 (*Left*) A cloud of scattered points in 3D. Points belong to one of four different regions: *red*, *green*, *blue*, and *grey*. (*Right*) The reconstructed surface obtained from the Voronoi interface of the functions defined in (7) and the mesh-generation algorithm

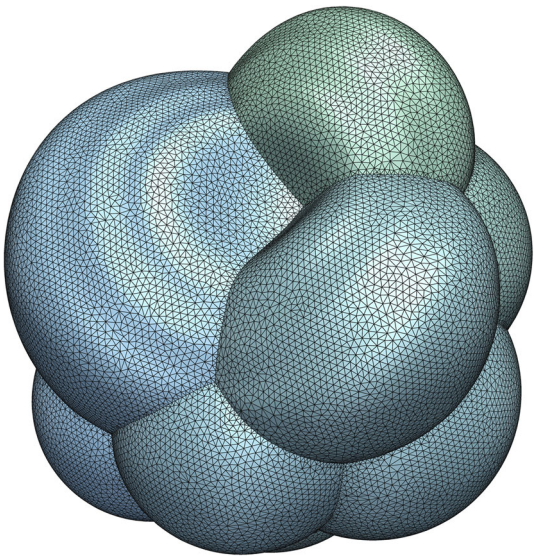

Fig. 11 A mesh of a cluster of soap bubbles, taken from a simulation of the multiscale dynamics of soap bubble foams [1]. The circular ripples seen near the top are capillary waves caused by the bubbles moving under the effects of surface tension. In this example, 20 iterations of force-based smoothing was used to generate the mesh, consisting of approximately 75,000 elements

generation algorithm to automatically create the meshes used in the finite element method. Moreover, it is imperative that the meshes be of high quality to accurately solve the underlying PDEs. The algorithm presented here was successfully used in this application: in a typical bubble simulation, tens of thousands of meshes are generated. Figure 11 shows a specific example in which circular ripples caused by capillary waves are seen near the top of the cluster.

6 Conclusions

In this paper, a robust and efficient mesh-generation algorithm for a network of interconnected surfaces has been presented. The algorithm capitalises on the notion of the Voronoi interface, which guarantees topological consistency of the created mesh and avoids heuristic or complex decisions about surface topology. An initial mesh of the Voronoi interface is created, which is then iteratively improved using mesh quality enhancement procedures. While the general approach can be implemented using any background tetrahedral tessellation, we have focused on an implementation that uses a BCC lattice together with vertex snapping—the BCC lattice creates an initial mesh with more uniformly distributed triangles, while the vertex snapping removes unnecessary sliver triangles. In the second stage, many possibilities exist for improving mesh quality—in this work, we have opted for forced-based smoothing, together with projection and edge flipping. The projection methods were designed to correctly maintain the accuracy of the surface representation while performing force-based smoothing. Combined with a clamping procedure, it was shown that small edges in the initial mesh finish expanding in as few as two to ten iterations, after which the remaining iterations are largely in improving mesh quality globally, where vertices spread apart due to the expansion factor in the rest edge length ℓ_0. It may be possible to analyse convergence more rigorously to establish quality bounds, similar to [19, 20, 21], although such analysis would be subtle, due to the intricate coupling of edge flipping, vertex movement, and clamping. In an extensive application of the meshing algorithm in [1], tens of thousands of meshes were automatically generated and

successfully used in finite element methods involving coupled boundary conditions at junctions. Possibilities for future work include investigating different force functions or clamping procedures, which could lead to very rapid convergence, and implementing a type of density control, allowing the mesh to be adaptively refined in some parts of the domain [29].

Acknowledgments This research was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, US Department of Energy, under contract number DE-AC02-05CH11231, by the Division of Mathematical Sciences of the National Science Foundation, and by an American Australian Association Sir Keith Murdoch Fellowship.

References

1. Saye RI, Sethian JA (2013) Multiscale modeling of membrane rearrangement, drainage, and rupture in evolving foams. Science 340(6133):720–724. doi:10.1126/science.1230623
2. Saye RI, Sethian JA (2011) The Voronoi Implicit Interface Method for computing multiphase physics. Proceedings of the National Academy of Sciences 108(49):19,498–19,503. doi:10.1073/pnas.1111557108
3. Osher S, Sethian JA (1988) Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. Journal of Computational Physics 79(1):12–49. doi:10.1016/0021-9991(88)90002-2
4. Saye RI, Sethian JA (2012) Analysis and applications of the Voronoi Implicit Interface Method. Journal of Computational Physics 231(18):6051–6085. doi:10.1016/j.jcp.2012.04.004
5. Lorensen WE, Cline HE (1987) Marching cubes: A high resolution 3d surface construction algorithm. Computer Graphics 21(4):163–169. doi:10.1145/37402.37422
6. Guéziec A, Hummel R (1995) Exploiting triangulated surface extraction using tetrahedral decomposition. IEEE Trans Visual Comput Graph 1(4):328–342. doi:10.1109/2945.485620
7. Payne BA, Toga AW (1990) Surface mapping brain function on 3d models. IEEE Comput Graph Appl 10(5):33–41. doi:10.1109/38.59034
8. Chan SL, Purisima EO (1998) A new tetrahedral tesselation scheme for isosurface generation. Computers and Graphics 22(1):83–90. doi:10.1016/S0097-8493(97)00085-X
9. Bloomenthal J, Ferguson K (1995) Polygonization of non-manifold implicit surfaces. In: proceedings of the 22nd annual conference on computer graphics and interactive techniques, SIGGRAPH '95, pp. 309–316. ACM. doi:10.1145/218380.218462
10. Suzuki H, Fujimori T, Michikawa T, Miwata Y, Sadaoka N (2007) Skeleton surface generation from volumetric models of thin plate structures for industrial applications. In: Martin R, Sabin M, Winkler J (eds.) Mathematics of Surfaces XII, lecture notes in computer science 4647:442–464. Springer Berlin Heidelberg. doi:10.1007/978-3-540-73843-5_27
11. Shammaa MH, Suzuki H, Ohtake Y (2008) Extraction of isosurfaces from multi-material CT volumetric data of mechanical parts. In: Proceedings of the 2008 ACM symposium on solid and physical modeling, pp. 213–220. ACM. doi:10.1145/1364901.1364931
12. Hege HC, Seebass M, Stalling D, Zöckler M (1997) A generalized marching cubes algorithm based on non-binary classifications. Tech. rep., Konrad-Zuse-Zentrum für Informationstechnik Berlin
13. Yamazaki S, Kase K, Ikeuchi K (2002) Non-manifold implicit surfaces based on discontinuous implicitization and polygonization. In: proceedings of geometric modeling and processing 2002, pp. 138–146. doi:10.1109/GMAP.2002.1027505
14. Pons JP, Ségonne F, Boissonnat JD, Rineau L, Yvinec M, Keriven R (2007) High-quality consistent meshing of multi-label datasets. In: Karssemeijer N, Lelieveldt B (eds.) Information Processing in Medical Imaging, IPMI'07, pp. 198–210. Springer Berlin Heidelberg. doi:10.1007/978-3-540-73273-0_17
15. Dey TK, Janoos F, Levine JA (2012) Meshing interfaces of multi-label data with Delaunay refinement. Eng Comput 28:71–82. doi:10.1007/s00366-011-0217-y
16. Boltcheva D, Yvinec M, Boissonnat JD (2009) Mesh generation from 3d multi-material images. In: Medical Image Computing and Computer-Assisted Intervention, MICCAI '09, pp. 283–290. Springer-Verlag. doi:10.1007/978-3-642-04271-3_35
17. Shimada K, Gossard DC (1995) Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing. In: Proceedings of the third ACM symposium on solid modeling and applications, SMA '95, pp. 409–419. ACM. doi:10.1145/218013.218095
18. Meyer M, Whitaker R, Kirby RM, Ledergerber C, Pfister H (2008) Particle-based sampling and meshing of surfaces in multimaterial volumes. IEEE Trans Visual Comput Graph 14(6):1539–1546. doi:10.1109/TVCG.2008.154
19. Bronson JR, Levine JA, Whitaker RT (2013) Lattice cleaving: Conforming tetrahedral meshes of multimaterial domains with bounded quality. In: Jiao X, Weill JC (eds.) Proceedings of the 21st International Meshing Roundtable, pp. 191–209. Springer Berlin Heidelberg. doi:10.1007/978-3-642-33573-0_12
20. Labelle F, Shewchuk JR (2007) Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. In: ACM SIGGRAPH 2007 papers, SIGGRAPH '07, vol. ~26. ACM, New York doi:10.1145/1275808.1276448
21. Chernikov AN, Chrisochoides NP (2011) Multitissue tetrahedral image-to-mesh conversion with guaranteed quality and fidelity. SIAM J Sci Comput 33(6):3491–3508. doi:10.1137/100815256
22. Reitinger B, Bornik E, Beichel R (2005) Constructing smooth non-manifold meshes of multi-labeled volumetric datasets. In: Proceedings of WSCG 2005, pp. 227–234. UNION Agency Science Press
23. Zhang Y, Qian J (2012) Resolving topology ambiguity for multiple-material domains. Comput Method Appl Mech Eng 247:166–178. doi:10.1016/j.cma.2012.07.022
24. Zhang Y, Hughes TJR, Bajaj CL (2008) Automatic 3d mesh generation for a domain with multiple materials. In: M.L. Brewer, D.Marcum (eds.) Proceedings of the 16th International Meshing Roundtable, pp. 367–386. Springer Berlin Heidelberg. doi:10.1007/978-3-540-75103-8_21
25. Liu Y, Foteinos P, Chernikov A, Chrisochoides N (2010) Multitissue mesh generation for brain images. In: Shontz S (ed.) Proceedings of the 19th International Meshing Roundtable, pp. 367–384. Springer Berlin Heidelberg. doi:10.1007/978-3-642-15414-0_22
26. Bloomfield MO, Richards DF, Cale TS (2005) The use of conformal voxels for consistent extractions from multiple level-set fields. In: Sunderam VS, van Albada GD, Sloot PA, Dongarra J (eds.) Computational Science - ICCS 2005, Lecture Notes in Computer Science, Vol. 3516, pp. 49–56. Springer Berlin Heidelberg. doi:10.1007/11428862_7
27. d'Otreppe V, Boman R, Ponthot JP (2012) Generating smooth surface meshes from multi-region medical images. Int J Numerical Methods Biomed Eng 28(6-7):642–660. doi:10.1002/cnm.1471
28. Anderson JC, Garth C, Duchaineau MA, Joy KI (2010) Smooth, volume-accurate material interface reconstruction. IEEE Trans Visual Comput Graph 16(5):802–814. doi:10.1109/TVCG.2010.17

29. Persson PO, Strang G (2004) A simple mesh generator in Matlab. SIAM Rev 46(2):329–345. doi:10.1137/S0036144503429121
30. Sethian JA (1996) A fast marching level set method for monotonically advancing fronts. Proceedings of the National Academy of Sciences 93:1591–1595. doi:10.1073/pnas.93.4.1591
31. Chopp DL (2001) Some improvements of the Fast Marching Method. SIAM J Sci Comput 23(1):230–244. doi:10.1137/S106482750037617X
32. Tsitsiklis JN (1995) Efficient algorithms for globally optimal trajectories. IEEE Transactions on Automatic Control 40(9): 1528–1538. doi:10.1109/9.412624
33. Adalsteinsson D, Sethian JA (1995) A fast level set method for propagating interfaces. J Comput Phys 118(2):269–277. doi:10.1006/jcph.1995.1098
34. Carr H, Möller T, Snoeyink J (2006) Artifacts caused by simplicial subdivision. IEEE Transactions on Visualization and Computer Graphics 12(2):231–242. doi:10.1109/TVCG.2006.22
35. Leng J, Zhang Y, Xu G (2012) A novel geometric flow-driven approach for quality improvement of segmented tetrahedral meshes. In: Quadros WR (ed.) Proceedings of the 20th International Meshing Roundtable, pp. 347–364. Springer Berlin Heidelberg. doi:10.1007/978-3-642-24734-7_19
36. Desbrun M, Meyer M, Schröder P, Barr AH (1999) Implicit fairing of irregular meshes using diffusion and curvature flow. In: Proceedings of the 26th annual conference on computer graphics and interactive techniques, SIGGRAPH '99, pp. 317–324. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. doi:10.1145/311535.311576
37. Persson PO (2005) Mesh generation for implicit geometries. Ph.D. thesis, Massachusetts Institute of Technology
38. Bern M, Plassmann P (1999) Mesh generation. In: Sack JR, Urutia J (eds.) Handbook of Computational Geometry. Elsevier Science, Amsterdam
39. Field DA (2000) Qualitative measures for initial meshes. Int J Numer Methods Eng 47(4):887–906. doi:10.1002/(SICI)1097-0207(20000210)47:4<887::AID-NME804>3.0.CO;2-H
40. Freitag LA, Jones MT, Plassmann PE (1999) The scalability of mesh improvement algorithms. In: Heath MT, Ranade A, Schreiber RS (eds.) Algorithms for parallel processing, The IMA volumes in mathematics and its applications, vol. 105, pp. 185–211. Springer New York. doi:10.1007/978-1-4612-1516-5_9