Optimized surface discretization of functionally defined multi-material objects

Pierre-Alain Fayolle and Alexander Pasko

The University of Aizu, Aizu-Wakamatsu city, Fukushima 965-8580, Japan

Bournemouth University, Poole, UK

Abstract

We present in this paper an algorithm for meshing implicit surfaces based on the Delaunay triangulation of a point-set adaptively sampled on an implicit surface. To improve the quality of the resulting triangular mesh, we use at each iteration a mesh optimization algorithm with the following objectives: optimizing the connectivity, retrieving the sharp features, regularizing the triangles shapes and minimizing the approximation error. Then, we extend this algorithm in order to handle functionally defined heterogeneous object surfaces, while maintaining a good quality for the triangles' shapes and the mesh features (geometrical sharp features and boundaries between different materials).

1. Introduction

Implicit surfaces [1,2] represent surfaces by continuous sets of points in space taking a given function value (isovalue), for example the surface defined by the isovalue 0 of \( f \) in \( \mathbb{R}^3 \) is: \( \partial \mathcal{S} = \{ \mathbf{p} \in \mathbb{R}^3 : f(\mathbf{p}) = 0 \} \). Replacing \( = \) by \( \geq \) in the previous equation defines more generally a solid instead of a surface. This representation is popular because of its ability to describe complicated shapes and to handle easily topological changes during simulation [3].

In this work, models for complex objects are built step by step in a constructive manner from simple primitives with predefined implicit surfaces by applying set operations (union, intersection, difference) to the primitives. Set operations can be implemented by using min/max or \( \mathbb{R} - \)functions (see for example [1,2,4]). The resulting complex objects are described by the zero level-set of a single function.

Visualization of such objects can be done by ray-tracing or by polygonization [5]. Polygonization consists in generating an approximation of the surface \( f = 0 \) by a discrete set of triangles. It is usually done by sampling the function on a regular grid and inspecting each cell in the grid to check if it intersects with the surface. Intersection between a cell and the surface is done by looking at the sign of the function values at each corner of the cell.

The problem we address in this work is to obtain an approximation of an implicit surface, which may be a boundary of a multi-material heterogeneous object, by a good quality optimized triangular mesh. The generated mesh should satisfy the following criteria: regularized shape and size of mesh triangles, optimized connectivity, retrieved sharp features, minimized approximation error (to the implicit surface) and mesh edges aligned with material boundaries for multi-material objects. Such meshes are required in object analysis using boundary-element and finite-element methods, object fabrication and other critical applications.

1.1. Related works

1.1.1. Marching cubes and variants

The marching cubes algorithm was first introduced by Lorensen and Cline in [5]. It is a popular algorithm for extracting a triangulated surface from a regular grid but suffers from several drawbacks: topological ambiguities, exhaustive enumeration of cells, badly shaped triangles or destroyed sharp features among others. A survey of the MC algorithm and its extensions was recently presented by Newman and Yi [6].

Natarajan [7] and Chernyaev [8] identify the problem of internal ambiguity and provide solutions. Lewiner et al. [9] provide an efficient implementation of Chernyaev [8]. Another approach for solving topological ambiguities is given in [10].

An algorithm for retrieving sharp features of the surface was proposed by Kobelt et al. [11]. Their algorithm extends the MC algorithm by using the local distance field information and its gradient to compute and insert into the mesh additional sample points...
lying on the surface’s features. Ohtake et al. [12,13] use a set of post-processing steps to improve the triangular mesh generated by the MC algorithm. Their work attempts to handle sharp features correctly as well as to regularize the mesh by vertex relocation.

In general, these algorithms tend to produce an excessive number of triangles, especially in the regions of low curvature, because of their use of a uniform grid. Moreover, the shapes of the triangles are not controlled and can degenerate even when a post-processing optimization is used.

Methods have been developed to apply the MC algorithm to an adaptive grid in [14,15]. However, cracks appearing when two cells of different resolutions meet need to be patched as done in [16].

The Dual Contouring algorithm introduced by Ju et al. [17] can also be used for extracting surfaces with adaptive resolution while reproducing sharp features (using Hermite data). The original Dual Contouring algorithm does not guarantee intersection-free surfaces and may have holes due to topological errors. It was later extended by Ju and Udeshi [18] to solve this issue. Ho et al. propose in [19] to convert the 3D marching cubes into 2D cubical marching squares. This algorithm generates surfaces adaptively without crack patching, maintains a consistent topology and sharp features. In these methods, there is no mechanism to control the shape of the triangles.

Improving the quality of marching cubes meshes is done in [20] by transforming the edges, intersecting the implicit surface, to places that improve the mesh quality. In [21,22], a new classification is presented for the MC case tables based on the different configurations of the edges intersecting the implicit surface within a cell. A modified MC case table is proposed based on this edge group formulation that yields significantly improved meshes compared to MC algorithms. Raman and Wenger propose a related approach in [23], however their method may generate non-manifold meshes and tends to change the mesh topology.

1.1.2. Isosurface approximation methods

Surface tracking methods start from a seed triangle on the surface and grow by iteratively adding new triangles to approximate the surface. Tracking algorithms can be traced to the work of Wyvill et al. [24]. In the Marching Triangles algorithm [25], the decision of adding a new triangle is based on the Delaunay surface constraint proposed by Boissonnat [26]. The original Marching Triangles algorithm suffers from cracks and is extended by Akkouche and Galin [27] to overcome this issue. In [28], Schreiner et al. propose an advancing front technique that uses a guidance field to generate sample on the surface. The choice of seed points is important for these tracking methods as there can be only as many connected components as seed points. In addition, none of these algorithms seem to be able to handle properly sharp features of the surface.

In the algorithm proposed in [29] by Desbrun et al., seed points on the bounding volume of an implicit surface migrate to the surface to generate a mesh approximating the surface. The algorithm is fast and allow interactive update of the mesh when adding or removing primitives. However, it is designed to work with implicit surfaces defined by skeletal elements only and it may fail to tessellate correctly regions where many primitives blend.

In [30], De Figueiredo et al. propose to sample the implicit surface with a particle system. Particles tend to be attracted by area of high curvatures. The sampling is then made uniform by introducing repelling forces between each particles and solving until equilibrium. An improved algorithm for sampling particles on an implicit surface is described by Witkin and Heckbert in [31]. A triangular mesh approximating the implicit surface is obtained from a Delaunay triangulation of the particles. Meyer et al. propose in [32] an algorithm that similarly samples particles on the implicit surface using a sizing field and then distributes them. The resulting point-set is then meshed using a Delaunay based algorithm. If the initial set of particles is not properly scattered, these algorithms may miss disjoint pieces and holes of the surface. These sampling approaches rely on heavy processing and are time consuming. Finally, sharp features are not handled properly by these algorithms.

1.1.3. Delaunay refinement

Chew describes in [33] an algorithm that constructs a restricted Delaunay triangulation of a surface by iteratively inserting the furthest intersection of surface with all Voronoi edges. Boissonnat and Oudot give in [34] an algorithm, which iteratively adds the centers of the bad surface Delaunay balls to the current point-set, until all Delaunay balls become good. Whether a surface Delaunay ball is good or not is decided based on the ratio of the circumsphere radius with a 1-Lipschitz function. In their analysis, the distance to the medial axis is used for the 1-Lipschitz function, which is rather difficult to compute practically. Dey et al. propose a related algorithm in [35] but the topological guarantee between the output mesh and the input surface is obtained by using a theorem by Edelsbrunner and Shah [36] on the topological ball property. They also employ techniques to limit the number of computations of the Delaunay triangulation. These methods do not capture sharp edges. Similarly to tracking methods, they also require some sample points on each connected component of the surface.

Gelas et al. propose in [37] an algorithm related to [35] with an additional post-processing step used for capturing sharp edges and regularizing the mesh following the re-meshing techniques of Valette and Chassery [38].

1.1.4. Polygonization of heterogeneous objects

The marching cubes algorithm has been extended by Nielson and Franken to handle multi-material instead of binary values in [39]. In their approach, each cell of a regular grid is first decomposed in tetrahedrons. They identify four cases of tetrahedrons (depending on the material index at each node) and show how to divide them to compute the material interface. Bloomenthal and Ferguson proposed another approach as part of their algorithm for polygonizing non-manifold implicit surfaces [40]. Suzuki et al. [41] proposed another extension of the marching cubes algorithm to handle multi-material objects by analyzing the possible patterns of \( n \) \( (n \in 1 \ldots 8) \) distinct materials in one cell. All these approaches are based on the marching cubes algorithm and therefore inherit its weaknesses. In their paper introducing the dual contouring algorithm [17], Ju et al. also describe an extension for multi-material modeling, which is similar to the approaches mentioned above [39–41]. This extension has naturally the same problems as the Dual Contouring algorithm. Shamma et al. introduce in [42] a method for the segmentation of multi-material CT data in order to extract boundary surface. In their approach, multi-material contouring is done by a technique similar to the one presented in [17].

Meyer et al. [43] recently extended the meshing algorithm proposed in [32] for meshing segmented 3D images. Their algorithm is designed for medical images and does not seem to handle sharp features. The initial sampling of the surface relies on heavy processing.

1.2. Overview and main contributions

In this paper, we present new algorithms for the generation of high-quality surface meshes from implicit surfaces. First a polygonization algorithm using adaptively sampled points on an implicit surface is proposed. Starting from a small sample of random points, mesh vertices are iteratively added where the surface approximation is insufficient, while the associated triangle mesh is optimized.

This algorithm is simple to implement, gives a direct control on the number of vertices on the mesh, handles sharp features and generates well shaped triangles. In contrast to marching cubes related
algorithms, the function is sampled only at the vertices and not at each cell corners (including empty cells away from the surface).

We extend this new algorithm for discretizing the surface of heterogeneous objects (objects made of multiple materials). Given a heterogeneous object defined by a set of functions (to define the object and material partitions geometries), we propose an algorithm that: (1) discretizes each material partition surface and (2) assembles them together while protecting the material boundaries. Resulting triangular meshes exhibit good quality while maintaining sharp features of the objects and boundaries between multiple materials.

2. Adaptive polygonization of implicit surfaces

In this section, we describe our polygonization algorithm based on the Delaunay triangulation of a set of points adaptively sampled on an implicit surface (see Algorithm 1). Details corresponding to each step are given in the following subsections.

Algorithm 1 computes a polygonal approximation of the surface corresponding to the zero level-set of a function \( f: \mathbb{R}^3 \rightarrow \mathbb{R} \). At first (lines 1–2), a small number of random points is created and each point is projected on the surface \( f = 0 \). In all our experiments, 5 random points are used to initialize the algorithm. We extend this point-set by additional points located at the nodes of regular grids on each face of a bounding box enclosing the initial random points (lines 3–4).

The main part of the algorithm is the loop in lines 5–16 where vertices are adaptively added to the triangle mesh and the mesh is optimized. A polygonal approximation of the surface is obtained by computing the Delaunay triangulation of the point-set \( P \), keeping the faces of the triangulation corresponding to the surface boundary \( f = 0 \) (lines 6–11). The mesh is optimized by a series of steps: extracting sharp features, regularizing the mesh and maintaining the vertices close to the surface (line 12). At each loop iteration, some of the tetrahedrons’ circumcenters (line 8) and triangles’ centroids (line 13) are selected, projected on the surface \( f = 0 \) and added to a list of possible additional vertices. If the number of vertices is still below the user specified number, we add extra vertices in area of high curvature (line 17–22).

Finally, the mesh is further optimized by a series of edge flipping and edge collapsing in order to improve the mesh quality (line 23).

Algorithm 1. Adaptive algorithm for the polygonization of an implicit surface

\[
\begin{align*}
\textbf{Require:} & \quad \text{A function } f: \mathbb{R}^3 \rightarrow \mathbb{R} \text{ and a maximum number of points } n. \\
1: & \quad \text{Generate a small number of random points } P = \{p_i\}. \\
2: & \quad \text{Project these points } p_i \text{ on the surface } f = 0. \\
3: & \quad \text{Generate a small number of ghost points } G = \{g_i\}. \\
4: & \quad \text{Add the ghost points to the original point-set: } P = P \cup G. \\
5: & \quad \text{while the number of mesh vertices is less than } n \text{ OR there is no more points to add do} \\
6: & \quad \text{Compute the Delaunay triangulation } D_t \text{ of the point-set } P. \\
7: & \quad \text{Remove the tetrahedrons from } D_t \text{ outside of the solid.} \\
8: & \quad \text{Select tetrahedron circumcenters } c_i \text{ such that } d(c_i)/r_i < \beta. \\
9: & \quad \text{Project } c_i \text{ on the surface } f = 0. \\
10: & \quad \text{Add these points to the list of additional points.} \\
11: & \quad \text{Find in } D_t \text{ the set of triangles } T \text{ on the surface.} \\
12: & \quad \text{Optimize the mesh } (T, P). \\
13: & \quad \text{Select triangle centroids } c_i \text{ such that } d(c_i) > h_o \text{ and } r_i > R_{\text{min}}. \\
14: & \quad \text{Project } c_i \text{ on the surface } f = 0. \\
15: & \quad \text{Add these points to the list of additional points.} \\
16: & \quad \text{end while} \\
17: & \quad \text{while the number of vertices is less than } n \text{ do} \\
18: & \quad \text{Compute the curvature at each triangle centroid.} \\
19: & \quad \text{Add the triangles in a priority queue (sorted by curvature).} \\
20: & \quad \text{Subdivide the first triangle in the queue using a one to four subdivision. Neighboring triangles are subdivided accordingly to avoid cracks and hanging vertices.} \\
21: & \quad \text{Add the newly generated triangles in the queue.} \\
22: & \quad \text{end while} \\
23: & \quad \text{Process triangles with a minimum angle below some user defined threshold by edge swapping/collapsing and vertex relocation (see Section 2.8).}
\end{align*}
\]

2.1. Projection on the implicit surface

Given a point \( p \) and a function \( f \), we are searching for the projection \( p_0 \) of the point \( p \) on the implicit surface \( f = 0 \). If \( f \) is the signed distance function, then the projection: \( p_0 = p - f(p)/\sqrt{f(p)} \) is exact (whenever the gradient of \( f \) is defined). If \( f \) is not the signed
distance function, it is possible to project the point on the surface using a numerical algorithm or to approximate the projection.

2.1.1. Projection for general implicit surface

\[ D_p = \frac{p_s}{C_0} \]

\( p_s \) is parallel to \( r_f(p_s) \) and at \( p_s \), \( f(p_s) = 0 \), leading to the following system of equations:

\[ f(p + \Delta p) = 0 \]
\[ \Delta p + t\nabla f(p + \Delta p) = 0 \quad (1) \]

By eliminating \( t \), we can get a system in \( p_s \) only:

\[ R(p_s) = \begin{pmatrix} f_x(p_s) \\ f_y(p_s) \\ f_z(p_s) \end{pmatrix} = 0 \]

\[ \begin{pmatrix} f_x(x_s - x) - f_x(y_s - y) \\ f_y(x_s - x) - f_y(z_s - z) \end{pmatrix} = 0 \quad (2) \]

where \( f_x, f_y \) and \( f_z \) are the partial derivatives of \( f \) with respect to \( x, y \) and \( z \) at \( p_s \).

Eq. (2) is solved for the unknown vector column \( p_s \) with the damped Newton method using as initial guess the current point \( p \); let \( J \) be the Jacobian matrix of the system \( R(p_{k+1}) = p_k - \alpha f(p_k)R(p_k) \) is iterated until \( R(p_k) < \epsilon \). The damping parameter \( \alpha \) is set to 1.0 in our experiments and \( \epsilon \) to \( 10^{-5} \). The system in Eq. (2) requires the function \( f \) to be smooth (\( f \in C^2 \)).

2.1.2. First order approximation

The previous projection can be time consuming. It is possible to obtain a good approximation of this projection. Starting with the first order Taylor expansion:

\[ f(p_s) = f(p) + D_p \nabla f(p) \]

and taking the gradient gives:

\[ \nabla f(p + \Delta p) \]

By combining with \( \Delta p + t\nabla f(p + \Delta p) = 0 \) (see Eq. (1)), we obtain: \( \Delta p + t\nabla f(p) = 0 \). Combined with \( f(p_s) = 0 \) we obtain:

\[ f(p) - t\| \nabla f(p) \|^2 = 0 \quad (3) \]

and:

\[ \Delta p = -f(p) \frac{\nabla f(p)}{\| \nabla f(p) \|^2} \quad (4) \]

The first order projection of \( p \) to the implicit surface \( f = 0 \) is thus given by:

\[ p - f(p) \frac{\nabla f(p)}{\| \nabla f(p) \|^2} \quad (5) \]

The above method works if \( \nabla f(p) \) exists and if \( \nabla f(p) \neq 0 \). For constructive models built using R-functions [1,4] this is the case except for some isolated set of points which are in general avoided by the algorithm since they are on the surface of the solid.

Practically, one iteration of Eq. (5) is not sufficient to reach the surface as illustrated by Fig. 1. We can iterate the projection given in Eq. (5) until \( f(p) < \epsilon \) (in our experiments we used \( \epsilon = 10^{-5} \)).
2.2. Additional ghost points

A small number of points located on a regular grid are added to the initial random points (lines 3–4). The regular grid is obtained by slightly enlarging in each directions the bounding box enclosing all mesh vertices and regularly adding points on each face of this grid (see Fig. 2 for an illustration in two-dimension).

We found experimentally that adding these points increases the quality of the resulting mesh. In all our experiments, we used 40 extra points in two-dimension and 600 extra points in three-dimension.

2.3. Removing external tetrahedrons

The Delaunay triangulation of a point-set covers the convex hull of the point-set. Tetrahedrons outside of the domain need to be peeled off (see Fig. 3 for an illustration in two-dimension). Distinguishing tetrahedrons inside and outside the volume to be meshed can be done by inspecting the sign of the function $f$ at the centroid $c$ of each tetrahedron: if $f(c) < 0$, then the tetrahedron is outside, otherwise it is inside.

Practically, we may want to keep a slightly thicker solid. One possible approach is to discard tetrahedrons for which $f(c) > -\epsilon$, for a small $\epsilon$ (we used $10^{-5}$ in all our experiments). Another approach is to compute the circumcenter $c$ and the circumradius $r$ of the tetrahedron $t$, and keep $t$ if it is outside and if $d(c)/r < \alpha$, where $\alpha$ is a user defined constant (smaller than 1) and $d$ is the distance from $c$ to the surface. If $f$ is not the signed distance function, then $d(c)$ can be approximated by $||c - c_i||$, where $c_i$ is the projection of $c$ on $f = 0$ and is computed as explained in section 2.1.

2.4. Additional tetrahedron circumcenters

We select tetrahedron circumcenters $c$ (line 8), where the ratio between the distance from $c$ to the surface and the circumradius $r$ is less than a given threshold (we used 0.2 in our experiments). This heuristic is trying to undersampled areas, which appear at the first iterations of the algorithm, when the number of points is small. Fig. 4 illustrates this type of configuration. The selected circumcenters are then projected on the surface and added to the list of additional points.

2.5. Mesh optimization

A triangular mesh approximating the surface $f = 0$ is computed in line 11. However, the triangles are not necessarily well shaped and the sharp features of the surface may not have been extracted correctly.

Optimization of the mesh is done by using an iterative scheme similar to the methods described in [13,12]. This scheme is summarized in Algorithm 2.

Algorithm 2. Mesh optimization

1: Move the triangle vertices to align the triangle normal with $\nabla f$ the gradient of $f$ (see Section 2.5.1).
2: Regularize the shape of the triangles by vertex relocation (see Section 2.5.2).
3: Project the vertices $p$ back on the surface $f = 0$ (see Section 2.1).

2.5.1. Sharp features extraction

There are various existing methods for extracting the sharp features of the object, one possibility is the method proposed by Ohtake et al. in [13], where mesh vertices are moved in order to align the triangle normals with the gradient of $f$. Other possible approaches consist in optimizing vertex positions using a quadric error metric similar to the one described in [44]. This approach was taken by Ju et al. in [17] and Ohtake et al. in [12].

Experimentally, we obtained good results with the method from [13] which we briefly summarize here. Let $n(T)$ be the normal to the triangle $T$, $m(x,y,z) = \nabla f(x,y,z)$ be the unit gradient to $f$, and $A(T)$ the area of the triangle $T$. The current vertex location $p$ is updated as follow:

$$p_{\text{new}} = p + \frac{1}{\sum_{T_i \in \text{inc}(p)} A(T_i)} \sum_{T_i \in \text{inc}(p)} A(T_i) n(T_i)$$

where $n(T_i) = \frac{m(T_i)}{||m(T_i)||}$, $m(T_i)$ is the projection of $m$ on the $m(T_i)$ direction, $c$ is the centroid of $T_i$, and $\text{inc}(p)$ corresponds to the set of triangles $\{T_i\}$ incident to the vertex $p$.

In addition to retrieving sharp features, this step is improving the location of the mesh vertices as well.

2.5.2. Mesh regularization

During the mesh regularization step, the modified position $p_{\text{new}}$ of a mesh vertex $p$ is obtained by:

$$p_{\text{new}} = \frac{1}{\sum_{T_i \in \text{inc}(p)} A(T_i)} \sum_{T_i \in \text{inc}(p)} A(T_i) c(T_i)$$

where $\text{inc}(p)$ corresponds to the set of triangles $\{T_i\}$ incident to the vertex $p$, $A(T_i)$ is the area of the triangle $T_i$, and $c(T_i)$ is the centroid of the triangle $T_i$.
To illustrate Eq. (7), consider a set of planar triangles around a vertex \( p \), then applying Eq. 7 to \( p \) will move this point to balance the area of each triangle. Especially, if the triangle vertices (other than \( p \)) lay on a circle, then \( p \) should be moved to the center of the circle as illustrated in Fig. 5.

Updating mesh vertices according to Eq. (7) may destroy sharp edges. It is preferable to identify sharp features and mark them as constraints, before invoking the mesh regularization procedure. Identifying sharp features is done by using a method proposed by Kobbelt et al. in [11]. A mesh vertex \( p \) is tagged as belonging to a sharp feature if: \( \min_i (\mathbf{n}_i, \mathbf{n}_j) < \alpha \), where \( \mathbf{n}_i \) is the gradient of \( f \) at the centroid \( c_i \) of the \( i \)-th triangle in the set of triangles incident to the vertex \( p \), and \( \alpha \) a user controlled parameter (we used a value of 0.7 for \( \alpha \) in our experiments).

2.5.3. Projection

Sharp feature extraction and mesh regularization may slightly move the mesh vertices away from the surface. In that case, the mesh vertices are projected back to the surface using the methods described in Section 2.1. The convergence of the methods described in section 2.1.1 or 2.1.2 is fast, since the vertices are initially already close to the surface.

2.6. Additional triangle centroids

We select a triangle centroid \( c \) (line 13), if its distance to the surface is greater than a given threshold and if the triangle circumradius \( r \) is greater than a given threshold (0.01 the minimum edge length of the bounding box in our experiments). Selecting triangle centroids in the area where the approximation error is high to properly adapt the surface approximation. Limiting the size of the circumradius allows to avoid too small triangles in the areas of high curvature. The selected triangle centroids are then projected on the surface and added to the list of points to be added.

2.7. Additional vertices

At the end of the main loop (lines 5–16) if the number of vertices is less than the user specified number \( n \), new vertices are added in areas of high curvature (lines 17–22). An indicator for the surface curvature is computed at each triangle’s centroid using the implicit surface function \( f \) (line 18). We compute this curvature indicator as: \( A_t |(\mathbf{n}_c - \mathbf{n}_1) + (\mathbf{n}_c - \mathbf{n}_2) + (\mathbf{n}_c - \mathbf{n}_3)| \) where \( A_t \) is the area of the triangle \( t \). \( \mathbf{n}_c = -\sqrt{f(c)} \) is the normal at the centroid \( c \).
of the triangle \( t \) (after projection on the surface) and \( n_i = \frac{\begin{vmatrix} \mathbf{c}_i \end{vmatrix}}{\| \mathbf{c}_i \|} \) are the normals at the centroids \( \mathbf{c}_i \) (after projection on the surface) of the triangles neighbors to \( t \) (i.e. the triangles sharing an edge with \( t \)). Triangles are then inserted in a priority queue and sorted by increasing curvature (line 19). The triangle with highest curvature indicator is extracted and subdivided using a one to four subdivision (line 20). Neighboring triangles to the subdivided triangle are handled accordingly to avoid cracks and hanging vertices at each iteration of the loop. A triangle adjacent to only one subdivided triangle is split in two. A triangle adjacent to at least two subdivided triangles is subdivided using one to four subdivision. Finally the newly created triangles are added to the priority queue. These steps are repeated until the number of vertices in the triangle is greater than the user specified number of vertices \( n \). After the addition of new vertices, we project again all vertices on the surface and optimize the triangle mesh with the techniques described in Section 2.5.

2.8. Mesh improvement

At line 23, the mesh may still contain a small number of skinny triangles with one or two very small angles. This will happen often near the surface sharp features.

An additional step is added to iterates through the triangles and either swap or collapse edges for triangles with angles below some user defined threshold (we used a minimum angle of 15 in all our experiments).

This step may affect the quality of the mesh. Especially, edges on sharp features can be swapped or collapsed during the process described above. In order to prevent this, we relocate the mesh vertices by minimizing a quadric error metric as described in [12]. We alternate these two steps (edge swapping/collapsing and mesh optimization) into a loop that stops when all triangles have a minimum angle above the user given threshold or if a maximum number of iterations has been reached (a maximum of 10 iterations was used in our experiments).

2.9. Discussion

The adaptive algorithm improves the first algorithm by adding new vertices where the geometric approximation is insufficient instead of starting with a fixed number of randomly distributed points. This implies that the adaptive algorithm should create fewer triangles in low curvature areas and more triangles in high curvature areas. Given the maximum number of vertices, it does a better usage of the vertices budget.

3. Discretization of heterogeneous objects

In this section, we describe a novel algorithm, based on Algorithm 1, to extract surface meshes from implicitly represented heterogeneous models. A heterogeneous object is an object made of different constituent materials. Our algorithm converts implicitly defined homogeneous objects into surface models separating the different homogeneous material regions.

3.1. Heterogeneous object modeling

Following the work described in [45], we model a heterogeneous object by an \((n+1)\)-tuple: \( M = (G, A_1, \ldots, A_n) \), where the geometry of the object \( G \) as well as the space partitions (or material regions) \( A_i \) for each material are defined by real valued functions: \( G = \{ x \in \mathbb{R}^3 : f(x) \geq 0 \} \), \( A_i = \{ x \in \mathbb{R}^3 : S_i(x) \geq 0 \} \), \( i = 1 \ldots n \). We add the condition that the regions \( A_i \) form a partition of the solid \( G \).

\[ G = \bigcup_{i=1}^{n} A_i \]  Each region has a unique material assigned to it: for \( i \neq j \) \( A_i \cap A_j \) is either empty or the surface boundary between \( A_i \) and \( A_j \).

We are interested in the discretization and the visualization of the heterogeneous object \( G \) surface with \( n \) materials assigned to the surface regions.

3.2. Algorithm for meshing of the heterogeneous object surface

Our meshing algorithm for heterogeneous objects contains two main steps: the first step is the meshing of each material region by using Algorithm 1 and the second step consists in assembling the meshes corresponding to each material region, while maintaining the detected boundaries between two adjacent materials. The pseudocode corresponding to this meshing algorithm is given in Algorithm 3.

**Algorithm 3.** Algorithm for meshing the surface of heterogeneous objects

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>for each space partition ( A_i (i \in {1 \ldots n}) ) do</td>
</tr>
<tr>
<td>2:</td>
<td>Compute a mesh ( (T_i, P_i) ) approximating the surface ( S_i = 0 ) with Algorithm 1.</td>
</tr>
<tr>
<td>3:</td>
<td>if ( A_i ) shares a boundary with another partition ( A_j (j &gt; i) ) then</td>
</tr>
<tr>
<td>4:</td>
<td>Compute the curve ( c_{ij} = \partial A_i \land \partial A_j \land \partial G ).</td>
</tr>
<tr>
<td>5:</td>
<td>Regularly sample ( c_{ij} ).</td>
</tr>
<tr>
<td>6:</td>
<td>Add these samples in a list of additional vertices ( L ).</td>
</tr>
<tr>
<td>7:</td>
<td>end if</td>
</tr>
<tr>
<td>8:</td>
<td>Remove the vertices from ( P_i ) which are not on the surface of ( G ).</td>
</tr>
<tr>
<td>9:</td>
<td>end for</td>
</tr>
<tr>
<td>10:</td>
<td>Compute ( B = \bigcup B_j ) the union of balls ( B_j = { (x, r) \mid x_j \in L } ).</td>
</tr>
<tr>
<td>11:</td>
<td>Compute ( P = P_i \cup \ldots \cup P_n ).</td>
</tr>
<tr>
<td>12:</td>
<td>Remove points from ( P ) inside ( B ) (see the discussion below).</td>
</tr>
<tr>
<td>13:</td>
<td>Compute the Delaunay triangulation ( D_i ) of ( P \cup X, X = { (x_j) } ).</td>
</tr>
<tr>
<td>14:</td>
<td>Remove tetrahedrons from ( D_i ), outside of ( G ).</td>
</tr>
<tr>
<td>15:</td>
<td>Identify ( T ) the faces of ( D_i ) on the surface of ( G ).</td>
</tr>
</tbody>
</table>

3.3. Discussion of the heterogeneous object discretization algorithm

A surface mesh for each partition (line 2) in Algorithm 3 is computing using Algorithm 1. This step produces good quality triangle meshes corresponding to the surface of each material \( \partial A_i = \{(x,y,z) : S_i(x,y,z) = 0 \} \). Meshing of each partition is independent tasks and can be run in their own thread.

The identification of the boundary curves between two materials (lines 3–4) is obtained from the functions defining the geometry of the object and the space partitions. Let \( S_i \) and \( S_j \) be the functions corresponding to the material regions \( A_i \) and \( A_j \). The boundary curve between the surfaces \( \{ p : S_i(p) = 0 \} \) and \( \{ p : S_j(p) = 0 \} \) can be defined by \( \{ p : F(p) = (\nabla S_i(p) \land \nabla S_j(p) = 0) \land \partial \text{intersection operation implemented by an R-function} \}. \) Determining if a vertex \( p \) belongs to the boundary or not is done by evaluating the function \( F(p) \) at that vertex. To improve the result, we can project each identified vertex along the gradient of \( F(p) \) on the boundary curve.

After the identification of vertices on the boundary curve between two materials, we identify the edges connecting these vertices in the original mesh and subdivide these edges by inserting additional vertices (line 5).

These additional vertices, as well as the original vertices on the boundary curves between two materials, are maintained in a list of
additional vertices (line 6). Points in this list are used as the centers of protecting balls (line 10). The union of these balls is formed (line 10) and the points corresponding to vertices of the surface meshes of each material that are inside this union of balls are discarded (lines 11–12). The radius of the balls $r$ controlling the distance between the centers can be chosen as the minimum length of the triangle edges over all the material meshes. It is also possible to refine the subdivision algorithm (line 5) by selecting a subdivision size locally: given an edge on the material boundary, we identify the triangles sharing this edge, and take for $r_i$ the minimum edge length for this triangle.

To improve the final result of Algorithm 3, it is possible to check if there is any edge in the final surface mesh that crosses the internal boundary between two materials, and project on the boundary the closer endpoint of the edge.

4. Experimental results

We present results that demonstrate the viability and effectiveness of our algorithms. Several examples of implicit surfaces and implicitly defined heterogeneous objects are meshed with our algorithms. All computed surface meshes are rendered using flat shading.

The algorithms presented in this work have been implemented in MATLAB. The computation of the Delaunay triangulation of a point-set is performed by Tetgen [46]. The computer used in the experiments was a Sun workstation, with an UltraSPARC IIIi processor at 1.5 GHz and 1 GB of main memory.

### 4.1. Results of the meshing algorithms

We first demonstrate that Algorithm 1 can produce surface mesh for complicated shapes as illustrated by Fig. 6. This object contains multiple disjoint components: the small sphere at the center of the object, the eight spherical parts around the core sphere and the outside torii. This object contains also several sharp features which can be seen on the right image in Fig. 6. The resulting surface mesh contains approximately 27,000 vertices.

Figs. 7 and 8 illustrate a torus and a more complex model meshed without adaptive selection of vertices (left) and with Algorithm 1 (right). Meshing without adaptive selection is done by generating

<table>
<thead>
<tr>
<th>Model</th>
<th>#v</th>
<th>#Tris</th>
<th>Min/average minimum angle</th>
<th>Error</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>3484</td>
<td>5764</td>
<td>28.3/52</td>
<td>1.4e-4</td>
<td>1.65</td>
</tr>
<tr>
<td>Torus</td>
<td>6815</td>
<td>12430</td>
<td>19.7/45</td>
<td>1.04e-4</td>
<td>2.9</td>
</tr>
<tr>
<td>Cylinders</td>
<td>13864</td>
<td>26528</td>
<td>20/46.5</td>
<td>1.25e-5</td>
<td>12.5</td>
</tr>
<tr>
<td>Core</td>
<td>26943</td>
<td>52678</td>
<td>16.9/46.3</td>
<td>3.6e-4</td>
<td>130</td>
</tr>
<tr>
<td>Sand</td>
<td>25856</td>
<td>50478</td>
<td>15.16/43</td>
<td>7.8e-4</td>
<td>170</td>
</tr>
<tr>
<td>Earth</td>
<td>30869</td>
<td>60606</td>
<td>18.92/43.5</td>
<td>7.1e-4</td>
<td>199</td>
</tr>
</tbody>
</table>
random points, projecting them on the surface (as described in Section 2.1) and applying the mesh optimization techniques described in Section 2.5. Adaptive insertion of vertices gives in general better results as illustrated in Fig. 8: vertex distribution is denser in areas of high curvature.

Table 1 shows quantitative results of Algorithm 1 on various models. The results are given in terms of the minimum and the average minimum angle over all triangles, the approximation error (computed as the sum of the squared distance error at each triangle centroid) and computation time (in seconds). The distance error at a point is approximated by:

\[ e(p) \approx \frac{|f(p)|}{|\nabla f(p)|} \]

where \( f \) is the implicit surface being meshed.

Table 2 shows quantitative results of Algorithm 1 on various models. The results are given in terms of the minimum and the average minimum angle over all triangles, the approximation error (computed as the sum of the squared distance error at each triangle centroid) and computation time (in seconds). The distance error at a point is approximated by:

\[ e(p) \approx \frac{|f(p)|}{|\nabla f(p)|} \]

Table 2

<table>
<thead>
<tr>
<th>Model</th>
<th>#\text{v}</th>
<th>#\text{Tris}</th>
<th>Min/average minimum angle</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macet cylinders</td>
<td>17308</td>
<td>34616</td>
<td>10.9/38.71</td>
<td>0.12</td>
</tr>
<tr>
<td>Macet core</td>
<td>22066</td>
<td>44120</td>
<td>17.16/40.23</td>
<td>0.019</td>
</tr>
<tr>
<td>Afront cylinders</td>
<td>6010</td>
<td>12020</td>
<td>13.8/49.67</td>
<td>0.15</td>
</tr>
<tr>
<td>Afront core</td>
<td>13659</td>
<td>27306</td>
<td>10.3/50.76</td>
<td>0.05</td>
</tr>
<tr>
<td>CGAL cylinders</td>
<td>3238</td>
<td>6476</td>
<td>30/45.9</td>
<td>0.0017</td>
</tr>
<tr>
<td>CGAL cylinders</td>
<td>82126</td>
<td>164248</td>
<td>29.99/45.9</td>
<td>2.4e-5</td>
</tr>
</tbody>
</table>

1 For interpretation of color in all figures, the reader is referred to the web version of this article.

We compare Algorithm 1 with existing algorithms for meshing implicit surfaces: macet [21,20,22], afront [28] and the algorithm discussed in [34]. Table 2 shows quantitative results of these algorithms applied to some of the previous models.

Macet and afront takes as input a three dimensional grid of scalar values. We used regular grids of 100 × 100 × 100 scalar values obtained by sampling the implicit surface functions. The low resolution mesh for the model made of cylinders was obtained by using the CGAL surface mesher [47] with an angular bound of 30 and a distance bound of 0.01. The mesh with higher resolution was obtained by using a distance bound of 0.001. None of these algorithms retrieve properly sharp features of the models as illustrated in Fig. 10. In comparison, our algorithm can generate meshes maintaining the sharp features of the original model while producing triangles of similar quality.

Finally, we demonstrate with Fig. 11 that our algorithm can mesh complex freeform models.

4.2. Results of the algorithm for the polygonization of heterogeneous objects

We show results of applying Algorithm 3 for meshing the surfaces of several heterogeneous objects. The first example is a cube.
made of three different homogeneous materials. The shape of each material region (space partition) is illustrated in Fig. 12. Material 1 is located in a half-torus (the torus is cut by the top face of the cube). Material 2 is defined in a volume bounded by a quarter of a sphere (the intersection of a sphere with three planes passing through its origin). Finally, the third material is located in the remaining space, i.e. the cube with subtracted space partitions of materials 1 and 2.

The final multi-material object is shown Fig. 13. Each material is assigned a different color: yellow for material 1, green for material 2 and blue for material 3. This mesh was generated in 45 seconds. It contains 6023 vertices and 10838 triangles. The approximation error (i.e. the sum of the squared error at each centroid) is $6.6 \times 10^{-5}$.

The second example is a chess pawn. The geometry of the pawn can be constructed by a set-expression given in [4]. Three materials have been defined over the pawn as illustrated in Fig. 14. The
second material region is defined by a complex geometry obtained from the intersection of a modified cylinder, where the height is bounded by a sinusoidal function, with the bottom part of the pawn. Different views and a zoom on the boundary curve are illustrated in Fig. 14. It took 172 seconds to generate this mesh. The sum of the squared error at each triangle centroid is 1.1e−22,298 triangles.

The final example, in Fig. 15, is made of two homogeneous materials defined on a sphere. The first material region, in yellow, corresponds to the union of several spirals. The second material region, in blue, corresponds to the rest of the sphere.

4.3. Limitations

The proposed algorithms rely on computing the Delaunay triangulation of a point-set. In our experiments, we used Tetgen [46], but other implementations exist and may be used as well such as the one available in CGAL [47].

One bottleneck of the algorithms is the number of evaluations of the function defining the solid (and the material regions). If the gradient of the function is unknown, it also has to be computed numerically by finite differences (which means three additional function evaluations). Function evaluation is needed when determining if a tetrahedron is inside or outside the solid, when projecting triangle centroids and tetrahedron circumcenters to the surface, and at each step during the mesh optimization (for retrieving sharp features and for projecting the vertices on the surface). In total, there are four function calls at each projection and there are three projections at each step (one projection for the triangle centroids, one projection for the tetrahedron circumcenters and one projection at each mesh optimization).

The triangles produced by our algorithm for meshing the surface of multi-material objects near the interfaces between two or more materials exhibit a lower quality than the rest of the triangles. This degradation is caused by the additional vertices added along the curved boundaries and necessary for keeping the edges of the triangle mesh aligned with the material boundaries.

5. Conclusion

We have proposed in this paper a new algorithm for meshing implicit surfaces that generate good quality triangle meshes. We have illustrated this algorithm by several practical examples, including complex implicit surfaces with several disconnected components and sharp features, and freeform objects. Using this algorithm, we have proposed a new algorithm for meshing surfaces of heterogeneous objects. This algorithm maintains the curved boundaries between different materials, while producing good quality triangle meshes.

Acknowledgments

We would like to thank the anonymous reviewers for their comments and suggestions. The models for ‘core’, ‘sand’ and ‘mouse’ are from the HyperFun project (http://www.hyperfun.org/wiki/doku.php?id=gallery:main).

References
