Technical Report 2009-002

# Polygonization of implicit surfaces using Delaunay triangulation

Pierre-Alain Fayolle

August 19, 2009

Computer Graphics Laboratory
The University of Aizu
Tsuruga, Ikki-Machi, Aizu-Wakamatsu City
Fukushima, 965-8580 Japan

**Technical Report 2009-002**

| |
|---|
| Title:<br>Polygonization of implicit surfaces using Delaunay triangulation |
| Authors:<br>Pierre-Alain Fayolle |
| Key Words and Phrases:<br>Polygonization, implicit surfaces, mesh generation, mesh optimization |
| Abstract:<br>We present in this report two algorithms for polygonizing implicit surfaces based on Delaunay triangulation. To improve the quality of the obtained triangle mesh, we propose a mesh optimization algorithm for: retrieving sharp features, regularizing mesh triangles, and minimizing the approximation error. We show how to combine an adaptive selection of vertices, with a Delaunay triangulation and mesh optimization to produce good quality triangle meshes. |

| | |
|---|---|
| Report Date:<br>8/19/2009 | Written Language:<br>English |

| |
|---|
| Any Other Identifying Information of this Report: |
| Distribution Statement:<br>First Issue: 10 copies |
| Supplementary Notes: |

**Computer Graphics Laboratory**
**The University of Aizu**
Aizu-Wakamatsu
Fukushima 965-8580
Japan

# Polygonization of implicit surfaces using Delaunay triangulation

Pierre-Alain Fayolle

07/15/2009

## Contents

**1 Introduction** **2**
1.1 Related works . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.2 Overview and main contributions . . . . . . . . . . . . . . . . . 5

**2 The first polygonization algorithm** **5**
2.1 Projection on the implicit surface . . . . . . . . . . . . . . . . 6
2.1.1 Projection for general implicit surface . . . . . . . . . 6
2.1.2 First order approximation . . . . . . . . . . . . . . . . . 7
2.2 Adding ghost points . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3 Removing external tetrahedrons . . . . . . . . . . . . . . . . . 8
2.4 Mesh optimization . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.4.1 Optimization of the connectivity . . . . . . . . . . . . 10
2.4.2 Sharp features extraction . . . . . . . . . . . . . . . . 10
2.4.3 Mesh regularization . . . . . . . . . . . . . . . . . . . . 11
2.4.4 Projection . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.5 Thin triangles removal . . . . . . . . . . . . . . . . . . . . . . . 12
2.6 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**3 Adaptive polygonization of implicit surfaces** **13**
3.1 Previous works . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
3.2 Main algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
3.2.1 Adding tetrahedron circumcenters . . . . . . . . . . . 15
3.2.2 Adding triangle centroids . . . . . . . . . . . . . . . . . 15
3.3 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

**4 Results and discussions** **16**

**5 Conclusion** **20**

**Abstract**

We present in this report two algorithms for polygonizing implicit surfaces based on Delaunay triangulation. To improve the quality of the obtained triangle mesh, we propose a mesh optimization algorithm for: retrieving sharp features, regularizing mesh triangles, and minimizing the approximation error. We show how to combine an adaptive selection of vertices, with a Delaunay triangulation and mesh optimization to produce good quality triangle meshes.

## 1 Introduction

In this report, we present two algorithms for the polygonization of implicit surfaces based on the Delaunay triangulation algorithm. Fig. 1 is illustrating some steps of the second algorithm.

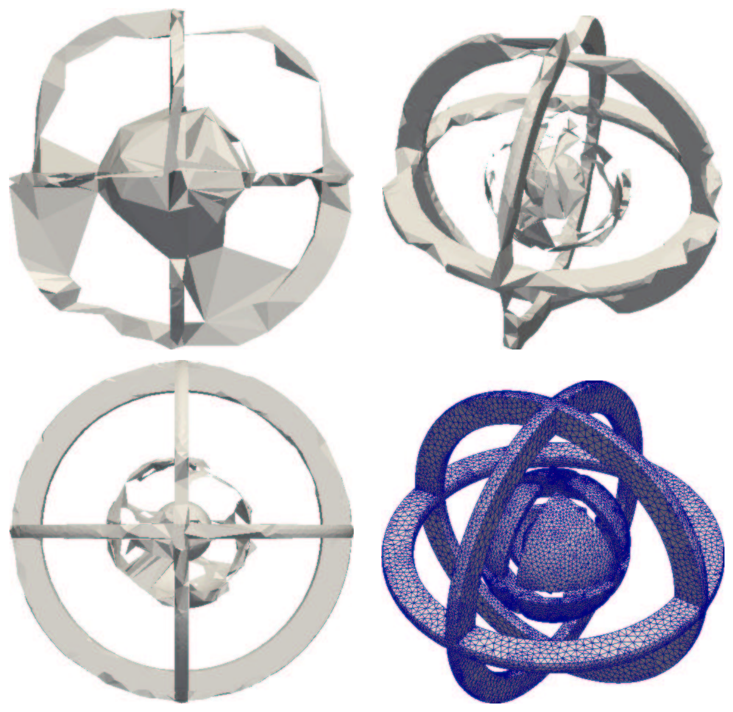

Figure 1: Some steps of the adaptive algorithm presented in section 3.

Implicit surfaces [6], or F-Reps [3] represent surfaces as the set of points in space taking a given function value (isovalue), for example the surface defined by the isovalue 0 of $f$ in $R^3$ is: $S = \{\mathbf{p} \in R^3 : f(\mathbf{p}) = 0\}$. Taking the inequality instead of the equality defines more generally a solid: $V = \{\mathbf{p} \in R^3 : f(\mathbf{p}) >= 0\}$.

This representation is interesting because of its ability to describe complicated shapes and handle easily topological changes during simulation [14].

Visualization is typically done by ray-tracing [11], i.e. shooting rays on a two dimensional grid and checking for their intersection with the object, or by polygonization [18, 4, 8]. Polygonization consists in generating an approximation of the surface $f = 0$ by a set of triangles. It is usually done by sampling the function on a regular grid and inspecting each cell to see if it intersects with the surface, which is done by looking at the sign of the values at each corner of the cell.

In this paper, we propose a new approach for the polygonization of implicit surfaces: random points are projected on the surface $f = 0$ and a tetrahedralization of the this point-set is generated by the Delaunay triangulation algorithm. The sign of the function is used to discard outside tetrahedrons. Triangles on the surface are extracted and their shape is improved by using information from the function and the current triangular mesh.

## 1.1 Related works

The Marching Cube algorithm [18] works by inspecting each cells of a regular grid and check for a potential intersection of the surface with the cell. An intersection is detected when the values at each corner of the current cell have different signs. A lookup table is used to retrieve the intersection topology based on the configuration of the signs in the cell. It is a popular algorithm for extracting a triangulated surface from a regular grid of values but suffers from several drawbacks such as: topological ambiguities, exhaustive investigation of cells, smoothing of the surface's sharp features, among others. For an in-depth review of the Marching Cube algorithm, its improvements and extensions, the reader is referred to the survey from Newman and Yi [17].

Pasko et al [4] proposed a similar algorithm, which also exhaustively inspects all cells in the grid, but is free of topological ambiguity in contrast to the original Marching Cube algorithm. The ambiguity in a cell is removed by noticing that the intersection of the implicit surface with the cell corresponds to the branches of an hyperbola.

An algorithm for retrieving the sharp features of the surface was proposed by Kobbelt et al. [12]. Their algorithm extends the Marching Cube 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 feature. Ohtake et al [19, 20] used a set of post-processing steps to improve the triangular mesh generated by the Marching Cube algorithm. Their work attempt to handle sharp features correctly as well as to regularize the mesh by relocating the mesh vertices based on the current mesh information and the implicit surface (function value and gradient).

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.

A different approach is the Dual Contouring algorithm introduced by Ju et al [15], which is designed for extracting surfaces with adaptive resolution (in contrast to using a uniform grid) while reproducing sharp features (based on normal information). However, the original Dual Contouring algorithm is not guaranteeing intersection-free surfaces, and it was later extended by Ju and Udeshi [16] to resolve this issue. In these methods, the size of the triangle mesh is controlled by the octree's depth. There is no mechanism in place to control the shape of triangles.

A different family of algorithm for the polygonization of implicit surfaces rely on surface tracking, i.e. these methods start from a seed triangle on the surface and grow by iteratively adding new triangles to approximate the surface. An example of surface tracking method is the Marching Triangles algorithm [2].

Starting from a seed triangle on the surface, the algorithm iteratively creates new triangles covering the surface by: adding a vertex in the plane of an existing triangle, projecting it to the implicit surface and deciding if the newly formed triangle should be added. The later decision is based on the Delaunay surface constraint proposed by Boissonnat [9].

The original Marching Triangles algorithm is not crack-free as noticed by Akkouche and Galin. They proposed to extend it by: improving the triangle creation step, especially the projection of the additional point on the surface, and to tessellate the cracks. Additionally, an incremental algorithm is proposed to integrate the improved marching triangles algorithm in an interactive implicit surface editing environment.

However, none of these algorithms seem to be able to handle properly sharp features of the surface.

Another type of algorithm was proposed by Desbrun et al [13]. Seed points on the bounding volume of an implicit surface migrate to the surface to generate a mesh approximating the surface. The algorithm is designed to work with implicit surfaces defined by skeletal elements only. The main strength of the algorithm is to be fast and to allow interactive update of the mesh when adding or removing primitives. This is due to the fact that the algorithm operates on elements locally and not globally on the implicit surface. The difficulty of the algorithm is to attach the triangles patches together. The algorithm may fail to tessellate correctly regions where many primitives blend. As mentioned earlier, it is also restricted to a single class of implicit surfaces: skeletal elements.

De Figueiredo et al. [7] proposed an algorithm inspired by physical systems to solve the problem of polygonization of implicit surfaces. First, a particle system is constrained on the implicit surface by solving a partial differential equation. Particles tend to be attracted by area of high curvatures. The sampling is then improved and made uniform by introducing repelling forces between each particles and solving until equilibrium. An improved algorithm for sampling particles on an implicit surface was presented by Witkin and Heckbert in [5]. Finally, a triangular approximation of the surface is obtained by keeping from the Delaunay triangulation of the point-set the triangles that approximate the tangent plane at each vertex. If the initial set of particles is not properly scattered, this algorithm may miss disjoint pieces and holes of the implicit surfaces. Scattering the points by solving the partial differential equation is time consuming, as is the regularization by introducing repelling forces. Finally, sharp features are not handled properly by the algorithm.

### 1.2 Overview and main contributions

In this paper, we present first a simple algorithm using Delaunay triangulation for the polygonization of an implicit surface $f$ given a vertex budget $n$. The algorithm consists in a few simple steps: 1) generate $n$ random points and project them on the boundary surface $f = 0$; 2) generate a set of tetrahedrons using the Delaunay triangulation algorithm; 3) discard outside tetrahedrons and generate triangles on the surface; 4) optimize the mesh (connectivity, surface approximation, triangles shape and sharp features). An adaptive version of the algorithm is then presented. Starting from a small sample of random points, it iteratively adds points in area of high curvature while optimizing the generated triangle mesh.

The presented algorithms are relatively simple to implement, give a direct control on the number of vertices on the mesh, handle sharp features and generate relatively well shaped triangles. In contrast, to marching cube algorithms, the function is sampled only on the vertices and not on every cells' corners (including in empty cells away from the surface). Additionally, the adaptive method provides a better distribution of the triangles' size by adding smaller triangles only in the high curvature areas.

## 2 The first polygonization algorithm

In this section, we describe the first version of the polygonization algorithm. The outline of the algorithm is given first. The details for each steps are given in the following subsections.

**Polygonize** (Polygonization of an implicit surface). Given a function $f : R^3 \to R$, a number of points $n$ and a rectangular box *bbox* bounding the space $\{(x, y, z) : f(x, y, z) \geq 0\}$, compute a polygonal approximation of the surface $\{(x, y, z) : f(x, y, z) = 0\}$.

1. [Generate points.] Generate $n$ points $\mathbf{p}$ in the bounded space $[bbox(1,1), bbox(2,1)] \times [bbox(1,2), bbox(2,2)] \times [bbox(1,3), bbox(2,3)]$. Points can be randomly generated or lying on a regular grid.
2. [Projection.] Project the points $\mathbf{p}$ on the surface $f = 0$.
3. [Add ghost points.] Generate a small number $m << n$ of ghost points $\mathbf{g}$ lying on regular grids, on each faces of a slightly enlarged box containing all the mesh vertices. Add the ghost points $\mathbf{g}$ to the original point-set $\mathbf{p}$: $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{g}$.
4. [Delaunay triangulation.] Compute the Delaunay triangulation $t$ of the point-set $\mathbf{p}$.
5. [Discard tetrahedrons.] Remove the tetrahedrons from $t$ outside of the volume $\{(x, y, z) : f(x, y, z) \geq 0\}$.
6. [Generate triangles.] Find the set of triangles $tri$ on the surface. A triangle is lying on the surface if it is not shared by tetrahedrons.
7. [Mesh optimization.] Optimize the triangular mesh ($tri$, $\mathbf{p}$) by repeating a set of simple steps that will optimize the connectivity, extract sharp features, regularize the mesh and maintain vertices close to the surface.
8. [Remove.] Remove triangles with an angle below some user defined threshold.

The algorithm (steps 1 to 6) is illustrated in two dimensions by Fig. 2. Steps 2, 3, 5, 7 and 8 are detailed in the following subsections.

## 2.1 Projection on the implicit surface

Given a point $\mathbf{p}$ and a function $f$, we are searching for the projection $\mathbf{p_s}$ of the point $\mathbf{p}$ on the implicit surface $f = 0$. If $f$ is the signed distance function, then the projection: $\mathbf{p} \leftarrow \mathbf{p} - f(\mathbf{p})\nabla f(\mathbf{p})$ is exact. If $f$ is not the signed distance function, it is possible to project the point on the surface by a numerical algorithm or to approximate its projection.

### 2.1.1 Projection for general implicit surface

$\Delta\mathbf{p} = \mathbf{p} - \mathbf{p_s}$ is parallel to $\nabla f(\mathbf{p_s})$ and at $\mathbf{p_s}$, $f(\mathbf{p_s}) = 0$, gives the following system of equations:

$$\begin{aligned} f(\mathbf{p} + \Delta\mathbf{p}) &= 0 \\ \Delta\mathbf{p} + t\nabla f(\mathbf{p} + \Delta\mathbf{p}) &= 0 \end{aligned} \tag{1}$$

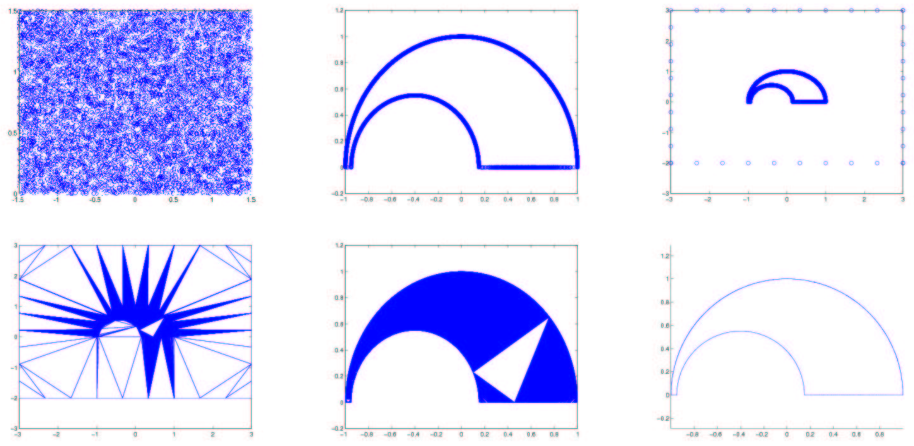

Figure 2: Illustration of the algorithm **Polygonize**, steps 1 to 6 in two dimensions.

By eliminating t, we can get a system in $\Delta \mathbf{p}$ only:

$$R(\mathbf{p_s}) = \begin{pmatrix} f(\mathbf{p_s}) \\ f_y(x_s - x) - f_x(y_s - y) \\ f_z(x_s - x) - f_x(z_s - z) \end{pmatrix} = 0 \tag{2}$$

where $f_x$, $f_y$ and $f_z$ are the partial derivatives of $f$ with respect to $x$, $y$ and $z$ at $\mathbf{p_s}$.

Eq. (2) is solved for the unknown vector column $\mathbf{p_s}$ with the damped Newton method using as initial guess the current point $\mathbf{p}$: let $J$ be the Jacobian matrix of the system $R$, $\mathbf{p}_{k+1} = \mathbf{p}_k - \alpha J^{-1}(\mathbf{p}_k)R(\mathbf{p}_k)$ is iterated until $R(\mathbf{p}_k) < \epsilon$.

### 2.1.2 First order approximation

The previous projection can be time consuming and we may want to avoid it. It is possible to obtain an approximation of the projection. Starting with a first order Taylor expansion: $f(\mathbf{p}_s) = f(\mathbf{p} + \Delta\mathbf{p}) \approx f(\mathbf{p}) + \Delta\mathbf{p}.\nabla f(\mathbf{p})$, and taking the gradient gives: $\nabla f(\mathbf{p} + \Delta\mathbf{p}) \approx \nabla f(\mathbf{p})$. By combining with $\Delta\mathbf{p} + t\nabla f(\mathbf{p} + \Delta\mathbf{p}) = 0$, we obtain: $\Delta\mathbf{p} + t\nabla f(\mathbf{p}) = 0$. $f(\mathbf{p}_s) = 0$ gives:

$$f(\mathbf{p}) - t\|\nabla f(\mathbf{p})\|^2 = 0 \tag{3}$$

and:

$$\Delta\mathbf{p} = \frac{\nabla f(\mathbf{p})}{\|\nabla f(\mathbf{p})\|^2} \tag{4}$$

The first order projection of $\mathbf{p}$ to the implicit surface $f = 0$ is thus given by:

$$\mathbf{p} \leftarrow \mathbf{p} - \frac{\nabla f(\mathbf{p})}{\|\nabla f(\mathbf{p})\|^2} \tag{5}$$

Practically, one iteration is not sufficient to reach the surface as illustrated by Fig. 3. We can iterate the projection (5) until $f(\mathbf{p}) < \epsilon$.

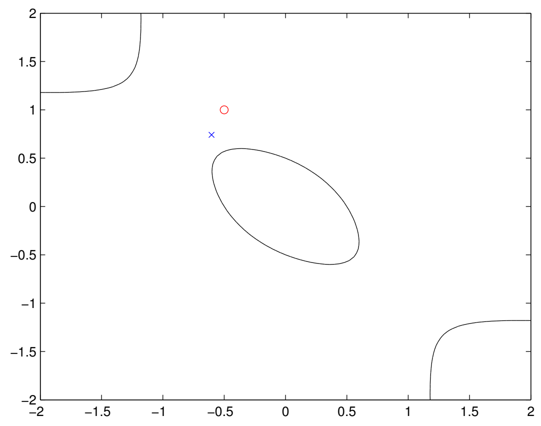

Figure 3: Error committed when projecting a point on an implicit surface with a first order projection. The original point is represented by the circle. The projected point is represented by the cross. The implicit curve is: $x^2 + y^2 + xy - 0.5x^2y^2 - 0.25$.

## 2.2 Adding ghost points

The step (3) of the **Polygonize** is adding a small number of points on a regular grid. The rectangular grid is obtained by slightly enlarging in each directions the rectangular box enclosing all mesh vertices (see Fig. 2). We found that adding these extra points increase the quality of the result.

## 2.3 Removing external tetrahedrons

The Delaunay triangulation of a point-set gives a triangulation of the convex hull of the point-set. Tetrahedrons outside of the domain need to be peeled off (see Fig. 4 for an illustration in two dimensions). Distinguishing between inside and outside tetrahedrons can be done by inspecting the sign of the function $f$ at the centroid $\mathbf{c}$ of each tetrahedron $t$: if $f(\mathbf{c}) < 0$, 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 tetrahedron for which $f(\mathbf{c}) < -\epsilon$. Another approach

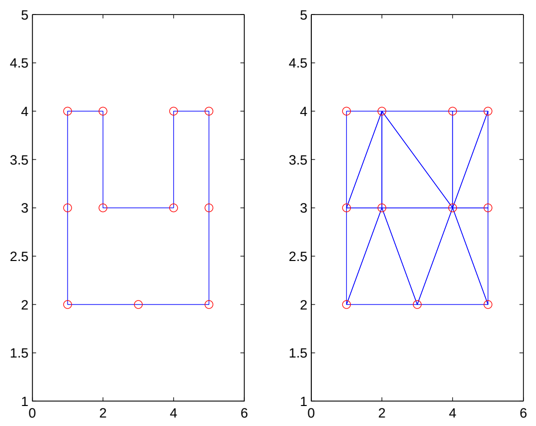

Figure 4: Delaunay triangulation of a point-set covers the convex hull. Outside tetrahedrons (triangles in two dimensions) need to be removed.

is to compute the circumcenter $\mathbf{c}$ and the circumradius $r$ of the tetrahedron $t$, and keep $t$ if it is outside and if $d(\mathbf{c})/r < \alpha$, where $\alpha$ is a user defined constant (smaller than 1) and $d$ is the distance from $\mathbf{c}$ to the surface. If $f$ is not the signed distance function, then $d(\mathbf{c})$ can be approximated by $\|\mathbf{c} - \mathbf{c}_s\|$, where $\mathbf{c}_s$ is the projection of $\mathbf{c}$ to $f = 0$ and is computed as explained in (2.1).

## 2.4 Mesh optimization

After the step (6) of the algorithm **Polygonize**, a triangular mesh approximating the surface $f = 0$ is obtained. However, the triangles are not necessarily well shaped and (in case the surface has sharp features) some sharp features may not have been extracted correctly.

Optimization of the mesh is done using an iterative scheme which resembles what was proposed by Ohtake et al [20]. This iterative scheme consists of the following steps:

**Mesh optimization** . Optimize the triangular mesh ($tri$, $\mathbf{p}$) by repeating a set of simple steps that will optimize the connectivity, extract sharp features, regularize the mesh and maintain vertices close to the surface:

- [Optimize connectivity.] if needed: recompute the Delaunay triangulation of $\mathbf{p}$, remove the external tetrahedrons and identify the triangles on the surface: $tri$.
- [Sharp features extraction.] Move the triangle vertices to align the triangle normal with $\nabla f$, the gradient of $f$.

- [Regularization.] Regularize the shape of the triangles.
- [Projection.] Project the points $\mathbf{p}$ on the surface $f = 0$ as in step 2 of **Polygonize**.

The first step modifies the connectivity of the triangle mesh while the other steps are vector fields $F_i$ that modifies the mesh vertices: $\mathbf{p}_{new} \leftarrow F_i(\mathbf{p}_{old})$.

### 2.4.1 Optimization of the connectivity

The mesh connectivity is optimized by computing the Delaunay triangulation of the point-set $\mathbf{p}$[1]. This is not needed for the first iteration of the mesh smoothing (since the mesh just got created), nor is it needed if the relative movement of each vertices is small (below a user defined threshold). Similar as in the triangle mesh creation, external tetrahedrons need to be removed (step 5) and triangles on the surface need to be extracted (step 6). This can be summarized by the following algorithm.

**Optimize connectivity** . Given a point-set $\mathbf{p}$, a triangular mesh $tri$, and the current iteration index $iter$. Let $\mathbf{p}_0 = \mathbf{p}$.

1. [Exit ?] if $\|\mathbf{p} - \mathbf{p}_{iter-1}\| < ttol$ exit.
2. [Delaunay triangulation.] Compute the Delaunay triangulation $t$ of the point-set $\mathbf{p}$.
3. [Discard tetrahedrons.] Remove the tetrahedrons from $t$ outside of the volume $\{(x, y, z) : f(x, y, z) >= 0\}$.
4. [Generate triangles.] Find the set of triangles $tri$ on the surface. A triangle is lying on the surface if it is not shared by tetrahedrons.

### 2.4.2 Sharp features extraction

For extracting the sharp features, we follow the method of Ohtake et al (section 2.2 in [20]): mesh vertices are moved in order to align the triangle normals with the gradient of $f$.

Let $N(\mathbf{p}, f)$ be the transformation applied to a mesh vertex $\mathbf{p}$, then the modified vertex $\mathbf{p}_{new}$ is set to: $\mathbf{p}_{new} \leftarrow \mathbf{p}_{old} + N(\mathbf{p}_{old}, f)$.

Let $n(T)$ be the normal to the triangle $T$, $m(x, y, z) = \frac{\nabla f(x,y,z)}{|\nabla f(x,y,z)|}$ be the unit gradient to $f$, and $A(T)$ the area of the triangle $T$. The transformation $N$ is defined by:

$$N(\mathbf{p}, f) = \frac{1}{\sum_{1ring(\mathbf{p})} A(T)} \sum_{1ring(\mathbf{p})} A(T)v(T) \qquad (6)$$

[1]the point-set contains the mesh vertices and the ghost points.

where $v(T) = (\overrightarrow{pc}.m(\mathbf{c})).m(\mathbf{c})$ is the projection of $\overrightarrow{pc}$ on the $m(\mathbf{c})$ direction, $\mathbf{c}$ is the centroid of $T$ and $1ring(\mathbf{p})$ is a function returning the set of triangles $T$ with $\mathbf{p}$ as a vertex.

As noted by Ohtake et al in [20], this step is actually doing more than only restoring sharp features: it also improves the position of the mesh vertices, as illustrated by the figure 5 in [20].

### 2.4.3 Mesh regularization

The modified position $\mathbf{p}_{new}$ of a mesh vertex $\mathbf{p}_{old}$ is obtained by:

$$\mathbf{p}_{new} \leftarrow \frac{1}{\sum_{1ring(\mathbf{p}_{old})} A(T)} \sum_{1ring(\mathbf{p}_{old})} A(T)\mathbf{c}(T) \tag{7}$$

where $1ring(\mathbf{p})$ is a function returning the set of triangles $T$ with $\mathbf{p}$ as a vertex, $A(T)$ is the area of the triangle $T$ and $\mathbf{c}(T)$ is the centroid of the triangle $T$.

Intuitively, if we consider a set of planar triangles around a vertex $\mathbf{p}$, then applying Eq. (7) to $\mathbf{p}$ should move it to balance the area of each triangles. Especially, if the triangle vertices (other than $\mathbf{p}$) lay on a circle, then $\mathbf{p}$ should be moved to the center of the circle as illustrated in Fig. 5.

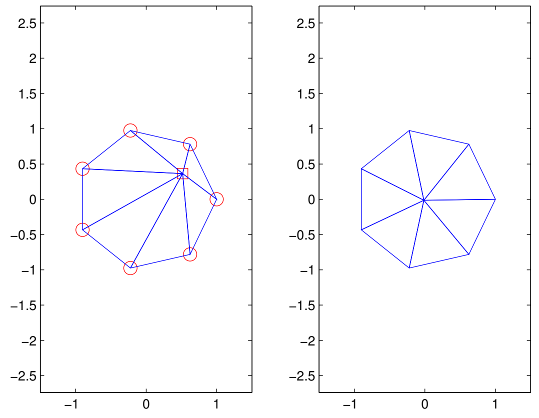

Figure 5: Illustration of the mesh regularization procedure Eq. 7. $p$ is moved to the center of the circle in order to balance the area of each triangle.

Updating mesh vertices according to Eq. (7) destroys sharp edges. It is preferable to identify sharp features and mark them as fixed, before invoking the mesh regularization procedure. Identifying sharp features is done following a method proposed by Kobbelt et al [12]. A mesh vertex $\mathbf{p}$ is tagged as belonging to a sharp feature if: $min_{i,j}(\mathbf{n_i}.\mathbf{n_j}) < \alpha$, where $\mathbf{n_i}$ is the gradient of $f$ at the centroid $c_i$ of the $i$-th triangle in the one ring of $\mathbf{p}$, and $\alpha$ a user controlled parameter.

#### 2.4.4 Projection

Eq. (7) tends to slightly move the mesh vertices away from the surface. The mesh vertices are projected back to the surface using the methods described in section 2.1. Even if the function $f$ is not a signed distance function, convergence of the methods 2.1.1 or 2.1.2 is practically fast, since the vertices are initially close to the surface.

### 2.5 Thin triangles removal

At the end of the step (7) of the algorithm **Polygonize**, it is possible that the triangular mesh contains some thin triangles with very small angles. This will most likely happen when the surface contains sharp features, because mesh vertices on sharp features are marked and fixed during the mesh regularization.

To resolve this issue, an additional step is added to iterate through the triangles and remove those with angle(s) below some user defined threshold. Removal is done by collapsing the edge opposite to the small angle and merging its incident vertices (see Fig. 6 top). If the triangle is flat with two angles below the threshold, we collapse the smaller of the two edges.

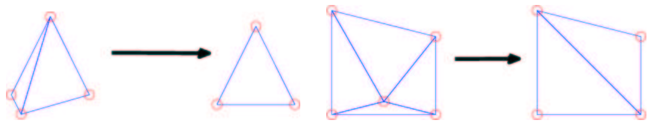
Figure 6: Removing thin triangles is done by collapsing edges opposite to small angles within a triangle. The top row illustrates the case where the triangle has one small angle and the bottom row illustrates the case where the triangle has two small angles.

### 2.6 Discussion

If the point-set is well sampled, then the polygonal approximation will be good (in term of topology and distance). The work of Amenta and Bern [1] characterizes good surface samplings. If the result of **Polygonize** is not satisfactory, it is always possible as a first and rough solution to increase the number of points $n$.

In term of complexity, steps (1) to (6) (included) have a worst case complexity of $O(n^2)$, because of the Delaunay triangulation. One iteration of the mesh optimization algorithm, step (7), has a worst case complexity

$O(n^2)$ (sharp features extraction and regularization are both $O(n)$). If the optimization is iterated $N$ times then the total worst case complexity is $O(Nn^2)$.

Algorithm **Polygonize** can be improved: the initial sampling of the $n$ points is random, and sharp features (if any) tend to attract points during the projection (see Figure). Even if the mesh optimization (especially step 2 and 3) tries to improve the vertices locations, there may still be too many triangles in area of low curvature and not enough in area of high curvature. It is possible to resolve that issue by: collapsing edges in area of low curvature and using an adaptive subdivision in area of high curvature. Another possibility explored in the next section and inspired by the work of Boissonnat and Oudot [10] is to use an adaptive algorithm.

## 3 Adaptive polygonization of implicit surfaces

In this section an adaptive version of the algorithm **Polygonize** is described. This algorithm extends the previous one by adding points in area insufficiently sampled, while maintaining an optimized mesh.

### 3.1 Previous works

Boissonnat and Oudot gave in [10] the theoretical background and an algorithm for the creation of a good surface sampling. Their algorithm proceed by iteratively adding the centers of the bad surface Delaunay balls to the current point-set, until all Delaunay ball become good. A surface Delaunay ball is defined in their work as any ball circumscribing a Delaunay facet $f$ and centered at the intersection of the surface with the Voronoi edge dual to $f$. Whether a surface Delaunay ball is good or not is decided based on the ratio of the circumradius with a 1-lipschitz function. In their analysis, the distance to the medial axis is used, which is rather difficult to compute practically.

In the following algorithm, the idea of adding points to the point-set is kept but the criteria for adding points is simplified. In addition, at each iteration, the current mesh is optimized (following the step (7) of **Polygonize** for a small number of iterations).

### 3.2 Main algorithm

**Adaptive polygonization** . Given a function $f : R^3 \rightarrow R$, a number of points $n$ and a rectangular box *bbox* bounding the space $\{(x, y, z) : f(x, y, z) \geq 0\}$, compute a polygonal approximation of the surface $\{(x, y, z) : f(x, y, z) = 0\}$.

1. [Generate points.] Generate $n$ points $\mathbf{p}$ in the bounded space $[bbox(1,1), bbox(2,1)] \times [bbox(1,2), bbox(2,2)] \times [bbox(1,3), bbox(2,3)]$. Points can be randomly generated or lying on a regular grid.

2. [Projection.] Project the points $\mathbf{p}$ on the surface $f = 0$.

3. [Add ghost points.] Generate a small number $m << n$ of ghost points $\mathbf{g}$ lying on regular grids, on each faces of a slightly enlarged box containing all the mesh vertices. Add the ghost points $\mathbf{g}$ to the original point-set $\mathbf{p}$: $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{g}$.

4. [Loop.] Repeat the following steps:

   (a) [Delaunay triangulation.] Compute the Delaunay triangulation $t$ of the point-set $\mathbf{p}$.

   (b) [Discard tetrahedrons.] Remove the tetrahedrons from $t$ outside of the volume $\{(x, y, z) : f(x, y, z) \geq 0\}$.

   (c) [Add tetrahedron circumcenters.] Select tetrahedron circumcenters $\mathbf{c}$ such that $d(\mathbf{c})/r < \beta$, where $d$ is the unsigned distance to the surface, $r$ is the circumradius and $\beta$ a user defined constant. Project $\mathbf{c}$ on the surface $f = 0$, and add to the list of additional points.

   (d) [Generate triangles.] Find the set of triangles $tri$ on the surface. A triangle is lying on the surface if it is not shared by tetrahedrons.

   (e) [Mesh optimization.] Optimize the triangular mesh ($tri$, $\mathbf{p}$) by repeating (for a small number of iterations) a set of simple steps that will: extract sharp features, regularize the mesh and maintain vertices close to the surface.

   (f) [Add triangle centroids.] Select triangle centroids $\mathbf{c}$ such that $d(\mathbf{c}) < h_0$ and $r < R_{min}$ where $r$ is the triangle circumradius, $h_0$ and $R_{min}$ are user defined constant. Project $\mathbf{c}$ on the surface $f = 0$ and add to the list of additional points.

   (g) [Exit ?] if the number of mesh vertices is greater than $n$ or there is no more points to add then exit the loop

5. [Mesh optimization.] Optimize the triangular mesh ($tri$, $\mathbf{p}$) by repeating a set of simple steps that will optimize the connectivity, extract sharp features, regularize the mesh and maintain vertices close to the surface.

6. [Remove.] Remove triangles with an angle below some user defined threshold.

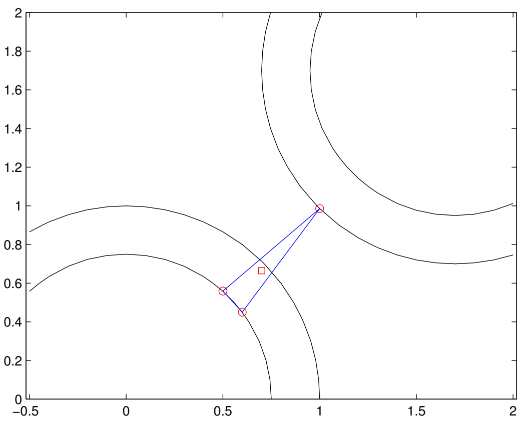

Figure 7: Because of an insufficient sampling during the first iteration of the algorithm, the surface may cross the inside of a tetrahedron.

Most of the components of this algorithm have already been discussed in section 2. Step (4.e) and (5) differ as no connectivity optimization is done in (4.e). There is no need to call the connectivity optimization in (4.e) because a Delaunay triangulation is already performed at each iteration of the loop (4) (in step (4.a)).

### 3.2.1 Adding tetrahedron circumcenters

Let $\mathbf{c}$ be the circumcenter of a tetrahedron $t$. $\mathbf{c}$ is selected if the ratio between the distance from $\mathbf{c}$ to the surface and the circumradius $r$ is less than a threshold. The goal of this rule is to try to capture undersampled areas, which appear at the first iterations of the algorithm, when the number of points is small. Fig. 7 shows the type of configuration that we want to capture. Selected circumcenters are then projected on the surface and marked as additional points.

### 3.2.2 Adding triangle centroids

Let $\mathbf{c}$ be the circumcenter of a triangle $tri$. $\mathbf{c}t$ is selected if its distance to the surface is greater than a given threshold and the circumradius $r$ is greater than a given threshold. Selecting triangle circumcenters in the area where the approximation error is high, allow to properly adapt the surface approximation. Limiting the size of the circumradius, allow to avoid too small triangles in area of high curvature. Selected triangle circumcenters are then projected on the surface and marked as additional points.

### 3.3 Discussion

The algorithm **Adaptive polygonization** improves the algorithm **Polygonize** by adding new vertices where the geometric approximation is insufficient instead of starting with a randomly distributed set of points. This means that **Adaptive polygonization** should allow fewer triangles than **Polygonize** in area of low curvature and more triangles in area of high curvature. Given a number of vertices $n$, it does a better usage of the vertices.

In term of complexity, the loop (4) for has a worst case complexity of $O(n^3)$ (in the case where one point is added at each iteration of the loop (4)). Step (5) has a worst case complexity of $O(Nn^2)$, where $N$ is the number of iterations in (5) (see section 2.6).

## 4 Results and discussions

It is possible to polygonize rather complicated shapes as illustrated by Fig. 8. This shape contains multiple disjoint components (the sphere in the middle, the eight spherical parts around the core sphere and the outside torii), and sharp features (see the right image in Fig. 8). The polygonal model in Fig. 8 was obtained by using the adaptive algorithm (section 3.2). 10 steps of the loop (4) were needed to reach the final sampling. 50 iterations of the mesh optimization were used to optimize the mesh quality. The mesh is made of 29,998 vertices and 59,980 faces. The average radius ratio[2] is 1.13 with a standard deviation of 0.17. The maximum ratio is 4.2.

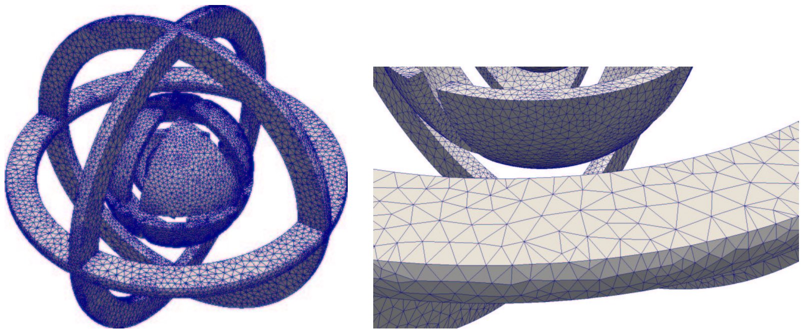

Figure 8: Polygonization of a complex model ('core') with several disjoint components, and sharp edges. See how sharp features are properly retrieved (right). The polygonal model is made of 29,998 vertices.

[2]the radius ratio is defined as the ratio of the radius of the circumscribing circle to the radius of the inscribed circle.

Both the algorithm presented in section 2, and the adaptive algorithm presented in section 3.2, produces good results, with good mesh quality (distributions of radius ratio and minimum angle are presented) as illustrated Fig. 9 (for the algorithm section 2) and Fig. 10 (for the adaptive algorithm section 3.2). The resulting polygonal models shown in Fig. 9 and 10 are made of approximately 7000 vertices.

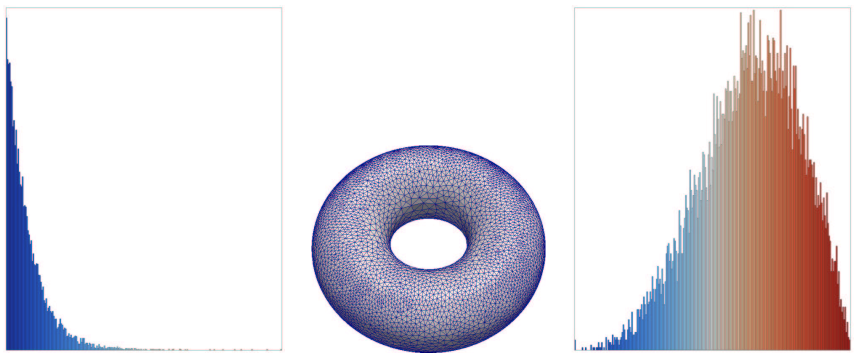

Figure 9: Result of the polygonization of a torus with the algorithm presented in section 2. Distributions of radius ratio and minimum angle illustrate the mesh quality.

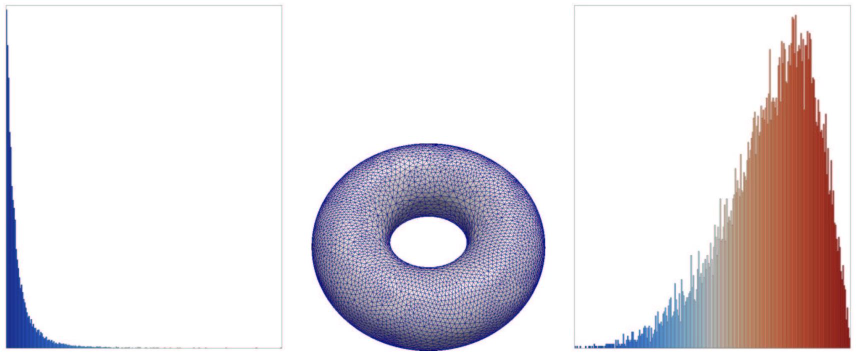

Figure 10: Result of the polygonization of a torus with the adaptive algorithm presented in section 3. Distributions of radius ratio and minimum angle illustrate the mesh quality.

In general, the adaptive algorithm does a better treatment of positioning the vertices as illustrated in Fig. 11. The model on the left has been created with the algorithm from section 2, while the model on the right has been created with the adaptive algorithm. The top row illustrates a view of the top of the object, which is flat. See how the adaptive algorithm adapts the vertex distribution to the zone of high curvature. On the other hand, algorithm **Polygonize** use lots of vertices on the flat zone, while area of higher curvature are more sparse (see Fig. 11 bottom left).

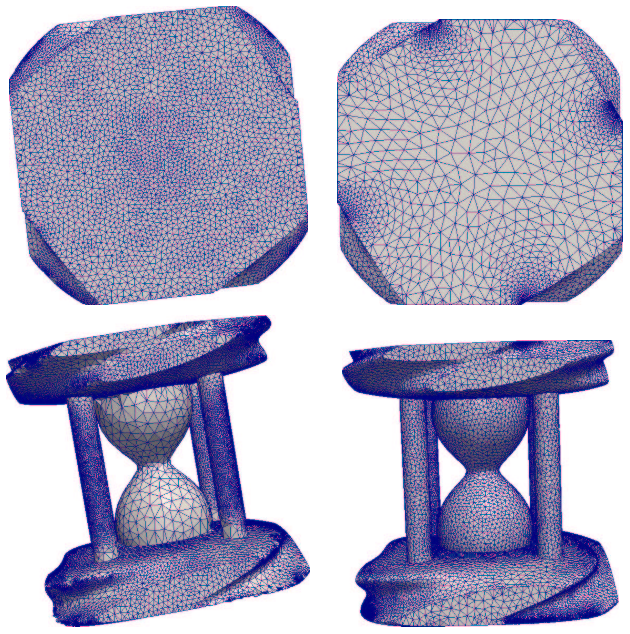

Figure 11: Illustration of the vertex distribution with the algorithm **Adaptive polygonization** (top and bottom rows, right column) against the vertex distribution with the algorithm **Polygonize** (top and bottom rows, left column) for the model 'sand'. See how vertices are concentrated on the area of high curvature with the adaptive algorithm.

Table 1 shows some qualitative results of the polygonization of several models using both the adaptive algorithm from section 3.2 and the algorithm **Polygonize** from section 2. For both algorithms, 50 iterations of the mesh optimization loop were performed.

**Effect of fixing vertices on sharp features during the mesh regularization** Before optimizing the position of each vertices according to eq. 7, we advised to identify vertices on sharp features and mark them, such that they remain fixed. Fig. 12 shows a cube polygonized with (left) and without (right) fixing the vertices on sharp features.

Fixing vertices on sharp features has a negative effect on the triangles shape, as it forbids the triangles around such vertices to be regularized. A practical approach is to allow the full mesh optimization, without fixing vertices, for some iterations, then allow to fix vertices on sharp features during the remaining iterations. See the results in fig. 13 where the left picture corresponds to 50 mesh optimization iterations with fixed vertices on

| Model | #v | #tris | max / average radius ratio | min / average minimum angle |
|---|---|---|---|---|
| sphere | 1000 | 1996 | 1.65 / 1.07 | 30 / 47.6 |
| sphere adaptive | 500 | 996 | 1.7 / 1.08 | 30 / 47.4 |
| torus | 6998 | 13996 | 2.2 / 1.09 | 25 / 46.4 |
| torus adaptive | 7219 | 14438 | 2.5 / 1.07 | 21 / 48.7 |
| cylinder | 3270 | 6538 | 6.04 / 1.18 | 15.11 / 44.5 |
| cylinder adaptive | 4467 | 8934 | 5.02 / 1.15 | 15.06 / 45.3 |
| core | 29396 | 58780 | 6.2 / 1.13 | 15.08 / 44.7 |
| core adaptive | 21769 | 43526 | 5.5 / 1.15 | 15.1 / 44.7 |
| sand | 27755 | 55540 | 7.45 / 1.13 | 15 / 45.5 |
| sand adaptive | 28103 | 56346 | 7.1 / 1.14 | 15 / 44.8 |

Table 1: Max/average radius ration and min/average minimum angle obtained on a series of models.

sharp features, while the right picture corresponds to 15 iterations without fixing vertices and 35 iterations with fixing the vertices, as they are reaching sharp features.

**Limitations** The limitations of the first algorithm (section 2) have been already mentioned in section 2.6 and addressed with the second algorithm (section 3).

Both algorithms rely on an implementation of the Delaunay triangulation algorithm. In our experiments, we used Matlab®, and its implementation of the Delaunay triangulation. The Delaunay triangulation is the bottleneck of the algorithm in terms of speed. A fast and robust implementation of Delaunay triangulation is required.

Both algorithms are not very fast. It is possible to speed up the adaptative algorithm by omitting the Delaunay triangulation during the mesh optimization (step 5). Practically, we did not find that removing the Delaunay triangulation from the mesh optimization, in step 5, had any noticeable effect on the final result. Figure 14 shows the minimum angle and radius ratio distributions for the model core with (left) and without (right) the Delaunay triangulation in step 5. On the other hand, for the algorithm **polygonize** (section 2), removal of the Delaunay triangulation in the mesh optimization results in a noticeable lower quality for the final mesh.

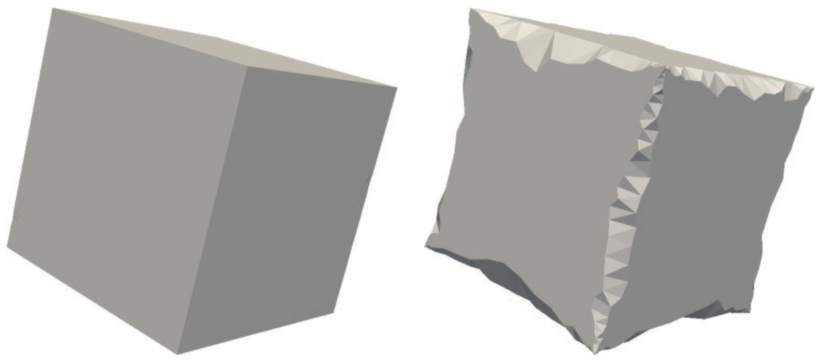

Figure 12: The impact of fixing vertices on sharp features is illustrated on the left; by allowing vertices to move following Eq. 7 sharp features are not retrieved properly (right) despite the sharp feature extraction step of section 2.4.2

The second bottleneck of the algorithms is the number of function (implicit surface) evaluations. If the gradient of the function is unknown, it has to be computed numerically by finite differences. In total, there are 4 function calls at each projection (there is one projection at each triangle mesh creation, and one projection at each mesh optimization). The gradient of the function is also needed in the mesh optimization for retrieving sharp features.

Both algorithms tend to produce triangles with one (or two) very small angle(s) close to sharp edges and corners. This is caused by the fact that vertices on sharp features are fixed during the mesh optimization. As discussed above, it is possible to first optimize the mesh without fixing vertices on sharp features, and then apply again the mesh optimization but with fixed vertices on sharp features. We also proposed to remove skinny triangles with small angles (see section 2.5). For the threshold angle, we used 15 degrees for model with sharp features (like for example: core, sand), and 20 degrees for others (like for example: sphere, torus).

## 5 Conclusion

We have proposed two algorithms for the polygonization of implicit surfaces based on Delaunay triangulation. We have illustrated the performance of the algorithms on several examples, including complex implicit surfaces with sharp features.

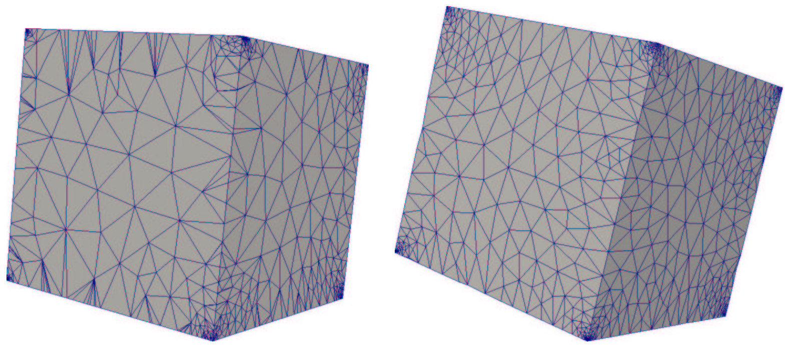

Figure 13: Early fixing of vertices on sharp features forbids good triangle shapes close to the sharp features. Left model was obtained with 50 iterations of the mesh optimization with vertices fixed on sharp features. Right model was obtained with 15 iterations of mesh optimization without locking vertices, and 35 iterations with locked vertices.

## References

[1] Amenta A. and Bern M. Surface reconstruction by voronoi filtering. In *ACM Symposium on Computational Geometry proceedings*, pages 39–48, 1998.

[2] Hilton A., Stoddart A., Illingworth J., and Windeatt T. Marching triangles : range image fusion for complex object modelling. In *International Conference on Image Processing proceedings*, 1996.

[3] Pasko A., Adzhiev V., Sourin A., and Savchenko V. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 8(11):429–446, 1995.

[4] Pasko A., Pilyugin V., and Pokrovskiy V. Geometric modeling in the analysis of trivariate functions. *Computers and Graphics*, 12:457–465, 1988.

[5] Witkin A. and Heckbert P. Using particles to sample and control implicit surfaces. In *ACM Siggraph 1994 proceedings*, pages 260–268, 1994.

[6] Jules Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan-Kaufmann, 1997.

[7] De Figueiredo L. H., De Miranda Gomes J., Terzopoulos D., and Velho L. Physically-based methods for polygonization of implicit surfaces. In *Graphics Interface 92 proceedings*, pages 250–257.

[8] Bloomenthal J. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–355, 1988.

[9] Boissonnat J.-D. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4), 1984.

[10] Boissonnat J.-D. and Oudot S. An effective condition for sampling surfaces with guarantee. Technical Report RR-5064, INRIA, December 2003.

[11] Hart J.C. Ray tracing implicit surfaces. Technical Report EECS-93-014, WSU, 1993.

[12] Kobbelt L., Botsch M., Schwanecke U., and Seidel H.-P. Feature sensitive surface extraction from volume data. In *SIGGRAPH'01 Proceedings*, pages 57–66.

[13] Desbrun M., Tsingos N., and Gascuel M.-P. Adaptive sampling of implicit surfaces for interactive modeling and animation. *Computer Graphics Forum*, 15(5):319–325, 1996.

[14] Osher S. and Fedkiw R. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002.

[15] Ju T., Losasso F., Schaefer S., and Warren J. Dual contouring on hermite data. In *ACM Siggraph 2002 proceedings*.

[16] Ju T. and Udeshi T. Intersection-free contouring on an octree grid. In *Pacific Graphics 2006 proceedings*.

[17] Newman T. and Yi H. A survey of the marching cubes algorithm. *Computers and Graphics*, 30:854–879, 2006.

[18] Lorensen W. and Cline H. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4), 1987.

[19] Ohtake Y. and Belyaev A. Dual/primal mesh optimization for polygonized implicit surfaces. In *ACM Solid Modeling proceedings*, pages 171–178.

[20] Ohtake Y., Belyaev A., and Pasko A. Dynamic mesh optimization for polygonized implicit surfaces with sharp features. *The Visual Computer*, 19(2-3):115–126, 2003.

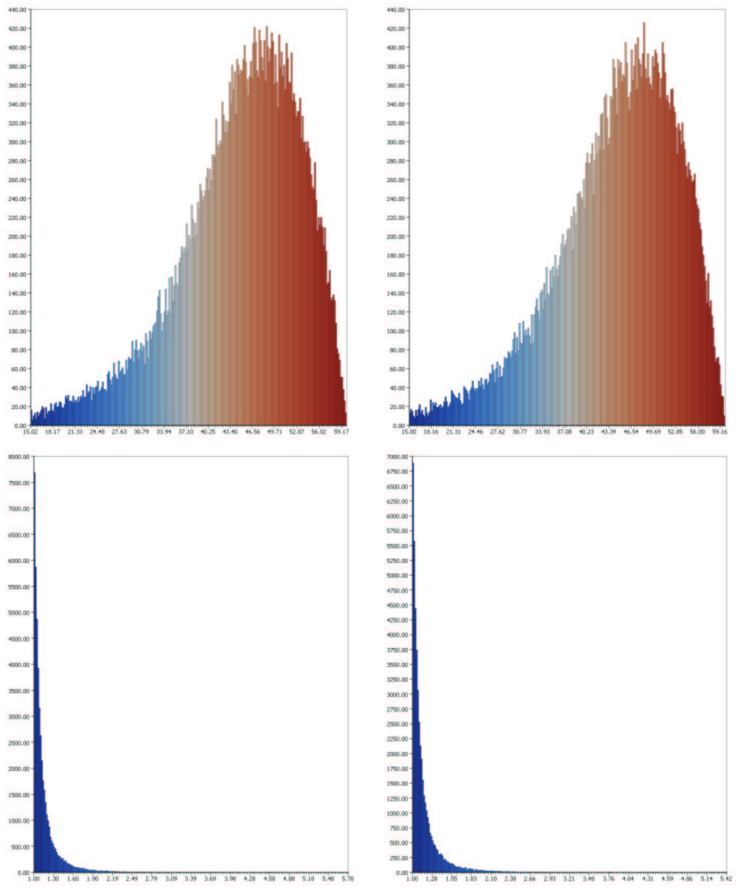

Figure 14: Minimum angle and ratio distributions for the model core obtained with the adaptative algorithm with (left) and without (right) Delaunay triangulation in the mesh optimization step 5. Similar distributions indicate that the effect of the Delaunay triangulation in the mesh optimization may not be so important.