Engineering with Computers
DOI 10.1007/s00366-012-0290-x

ORIGINAL ARTICLE

Lloyd's energy minimization in the L_p norm for quadrilateral surface mesh generation

Tristan Carrier Baudouin · Jean-François Remacle · Emilie Marchandise · Jonathan Lambrechts · François Henrotte

Received: 30 December 2011 / Accepted: 25 September 2012

Abstract Indirect methods recombine the elements of triangular meshes to produce quadrilaterals. The resulting quadrilaterals are usually randomly oriented, which is not desirable. However, by aligning the vertices of the initial triangular mesh, precisely oriented quads can be produced. Levy's algorithm is a non-linear optimization procedure that can align points according to a locally defined orientation matrix. It minimizes an energy functional based on the L_p distance. The triangulation of a set of vertices smoothed with Levy's algorithm is mainly composed of right-angled triangles, which is ideal for quad recombination. An implementation of Levy's algorithm based on numerical integration is presented. The implementation has the advantage of not modifying the edge meshes. It also features automatic calculation of the orientation angle. When used in combination with an indirect recombination algorithm, it can create quads of varying size and orientation. It has been tested on two-dimensional surfaces as well as globally parametrized three-dimensional surfaces. The results demonstrate an increase in the number of nodes having four neighbors and an improvement of the quads quality. The development took place in the framework of the Gmsh free software.

Keywords Clipped Voronoi diagrams · Centroidal Voronoi tessellations · Non-linear optimization · Quad mesh generation

T. C. Baudouin (✉) · J.-F. Remacle · E. Marchandise · J. Lambrechts · F. Henrotte
Institute of Mechanics, Materials and Civil Engineering, Université catholique de Louvain, Avenue Georges-Lemaître 4, 1348 Louvain-la-Neuve, Belgium
e-mail: tristan.carrier@uclouvain.be

1 Introduction

For finite element analysis, quad meshes can be advantageous compared to triangular meshes. For example, in computational fluid mechanics, they accelerate grid convergence [15, 23, 24] and they capture boundary layers with a higher precision [5]. They are also very useful in structural mechanics. They are not subject to numerical locking [18] and they allow schemes to remain stable under inexact integration [7, 27, 28]. In the context of high order methods, quad meshes can be curved more robustly [19].

However, quad mesh generation techniques are not as mature as triangular ones. The indirect approach looks like a promising solution. It consists of combining two by two the elements of a triangular mesh in order to create quads. Triangular mesh generators are usually designed to produce near-equilateral triangles. Combining these triangles yields randomly oriented quads, which is not ideal. Nevertheless, if the vertices of the initial triangular mesh are well aligned, an indirect algorithm like Blossom-Quad [17] can create high quality, precisely oriented quads. Unlike triangles, quads are always oriented. It is usually desirable to prescribe this orientation.

Lloyd's algorithm is a well-known procedure used to create uniform distributions of points. It iteratively moves each point to the centroid of its Voronoi cell [4]. The shapes of the Voronoi cells become as close as possible to circles. It can be shown that Lloyd's algorithm minimizes an energy functional equal to the sum of the moments of inertia of the Voronoi cells [11]. Some authors have used the limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) optimization procedure in order to minimize the energy functional [11]. LBFGS has the advantage of converging faster than the traditional fixed-point iteration process [11].

Published online: 09 October 2012

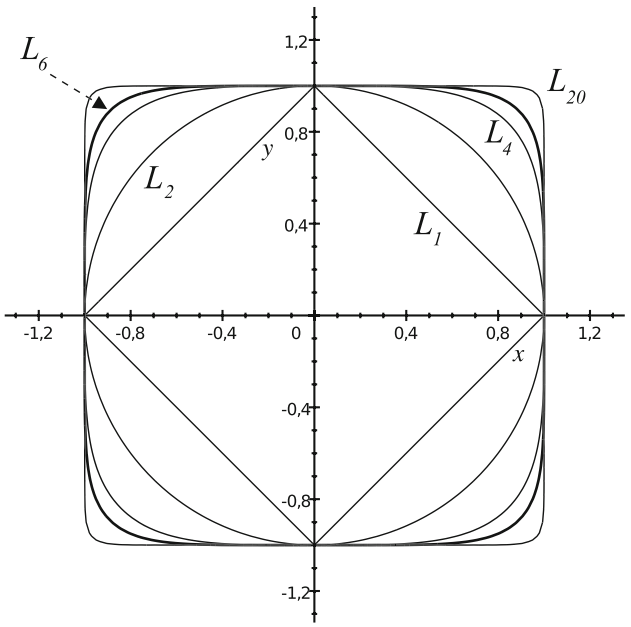

Fig. 1 Unit circles in various L_p distances

Levy's algorithm also minimizes the sum of the moments of inertia of the Voronoi cells [10]. However, the moments of inertia are now evaluated with the L_p distance instead of the classical Euclidean one. The Voronoi cells become rectangular, which has the effect of aligning the points in a rectangular manner.

Various L_p distances are illustrated on Fig. 1. They can be defined as [10]:

$$||\boldsymbol{y} - \boldsymbol{x}||_p^p = |y_1 - x_1|^p + |y_2 - x_2|^p \quad (1)$$

The L_p distances are only used for the energy functional. The Voronoi cells are still constructed with the habitual Euclidean distance.

Each curve on Fig. 1 contains a set of points equidistant to the origin according to its particular L_p distance. The curves become more and more rectangular as p increases. For any p different than two, the L_p distance becomes anisotropic.

Levy's algorithm also uses the LBFGS procedure in order to minimize the energy functional. LBFGS only requires the value of the functional and its gradient. However, the second derivative of the functional needs to be continuous. It has already been proven that the L_2 energy functional is C^2 continuous at certain conditions [11]. Nevertheless, there is currently no proofs available for the general L_p case.

The process is illustrated on Figs. 2 and 3. Figure 2a is the initial triangular mesh and Fig. 2b is the Voronoi diagram of the vertices. Figure 3a is the final quad mesh obtained with Levy's and Blossom-Quad algorithms. Figure 3b is the corresponding Voronoi diagram.

A two-dimensional implementation of Levy's algorithm adapted to the field of finite element computation is presented

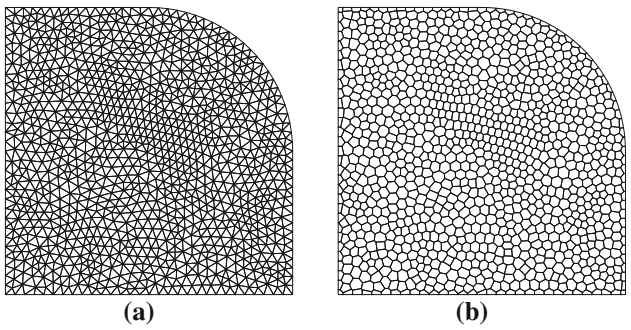

Fig. 2 The initial triangular mesh (**a**) and its corresponding Voronoi diagram (**b**)

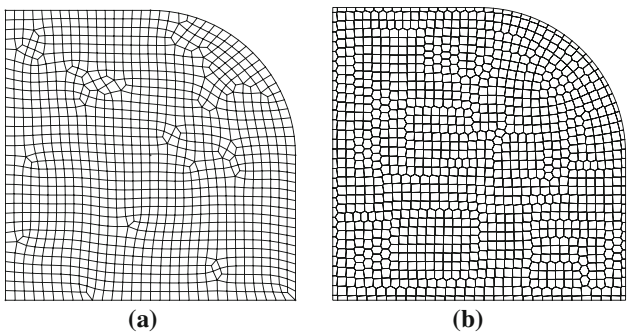

Fig. 3 The final quad mesh (**a**) and its corresponding Voronoi diagram (**b**) (a Laplacian smoothing has been applied after the triangles recombination)

in this paper. The implementation does not modify the one dimensional meshes discretizing the model edges. It can take into input angle and mesh size fields. These fields define the orientation and the size of the quads. The angle field is calculated in a completely automatic manner. The algorithm can be applied to any curved surface having a conformal parametrization. When no conformal parametrization exists, it is possible to build one using global parametrization techniques [14]. The energy and the gradient are computed with well-known Gauss integration techniques. The Alglib library [2] is employed for the LBFGS optimization part.

This article is divided as follows. Section 2 briefly discusses some methods that can align points in a rectangular manner. Section 3 details Levy's algorithm. Section 4 describes the algorithm used to compute the orientation angle. Section 5 presents several examples of quad meshes on three-dimensional surfaces.

2 Previous work

This section begins by presenting three methods that can align points in a rectangular manner. In order to create quad meshes, a recombination module needs to be supplemented.

A two-dimensional square packing heuristic is described in [20] and a three-dimensional one in [25]. These algorithms are inspired from the field of newtonian molecular dynamics. Each particle has a potential, which give rise to an attraction-repulsion force. The resulting equation of motion is solved using a Runge–Kutta scheme. The number of points is variable. It can either increase or decrease depending on the population density. In two dimensions, a one step refinement is used in order to create all quad meshes.

Another two-dimensional method based on potential fields is detailed in [22]. The distances are computed with the L_∞ norm instead of the usual Euclidean norm. Additional torsion spring-like forces are also used. The number of points is adjusted by splitting or collapsing edges. Special precautions are taken in order to avoid splitting an edge that was just collapsed or vice-versa. This method is able to produce quad meshes of high quality and high anisotropy on plane surfaces.

A two-dimensional procedure based on forces between neighboring vertices is described in [9]. Three different forces are used: two for alignment and one for uniform distribution. In order to fasten convergence, certain edges are aggregated together when they are sufficiently well aligned. Some quads are also split in order to reduce the number of T-vertices, i.e. vertices that have three neighbors. By using a local parametrization, this method is able to create high quality quad meshes on three-dimensional surfaces. However, it also produces a limited number of polygons with more than four faces.

The two following approaches to align points come from the field of digital art. They were initially developed for mosaic tilings but could be useful for quad mesh generation.

The Hausner method [6] is based on the classical Lloyd's algorithm, which iteratively moves vertices to the centroid of their Voronoi cells. However, this time the Voronoi diagrams are computed with the L_1 norm instead of the Euclidean norm. The method is therefore able to align points in precise directions instead of simply distributing them uniformly. The Voronoi diagrams have also the particularity of being discrete and not continuous. They are created by projecting pyramids on the graphic card Z-buffer. The overall procedure is very simple, which is a great advantage. Unfortunately, it may not be practical for meshes with large variation in element size because of the limited number of pixels.

In [8], the authors define an energy based on the L_1 distance. The energy is then approximated as a spring potential and an equation of motion is derived. A local parametrization allows the treatment of three-dimensional surfaces. In order to improve the final solution, certain tiles from high-density areas are moved to low-density areas. Well-aligned tilings of several thousand elements can be generated on curved surfaces.

So far, parametrization methods have not been discussed in this section. These methods differ completely from the precedent ones and do not rely on moving mesh vertices. For example, the procedure discussed in [21] subdivides a three-dimensional surface into a limited number of simple domains by using its cross-field information. These domains are then parametrized as rectangles and a full quad mesh of very high quality is generated. In [12], Lagrangian optimization is used to compute a smooth cross field. A parametrization that respects the cross field is later obtained with a mixed integer solver. Again, the quality of the final mesh is very high, even for complex surfaces.

The last method to be presented in this section comes from the field of image processing [13]. It aims at converting an image into a segmented mesh. The quads are created by combining triangles in no particular order. Convexity is the only requirement. The quads and the remaining triangles are then split in order to generate a full quad mesh. An original smoothing scheme based on Bézier control points is also used.

3 Levy's algorithm

The Voronoi cells of the mesh vertices that belong to the convex hull extend to infinity. The clipped Voronoi diagram is the part of the Voronoi diagram that is inside the domain, as shown in Figs. 2b and 3b. It is indispensable in order to compute accurate values for the energy and the gradient. The method used in this paper to generate clipped Voronoi diagrams is inspired from [26] and is explained in Sect. 3.1. Section 3.2 shows how to compute the energy and its gradient. Section 3.3 discusses the various difficulties related to the optimization step.

3.1 Clipped Voronoi diagrams

Figure 4 shows an example of a clipped Voronoi diagram.

Both boundary and interior vertices have Voronoi cells. Most of the time, it is only the Voronoi cells generated by boundary vertices that intersect the boundary. However, if an interior vertex is too close to a boundary, its Voronoi cell might intersect it, as shown in Fig. 5. This is unlikely, but the implementation should take this phenomenon into account.

Section 3.1.1 gives some results about the convexity of the clipping polygons. Section 3.1.2 presents a method that can identify all Voronoi cells that intersect the boundary. Section 3.1.3 explains the global clipping algorithm.

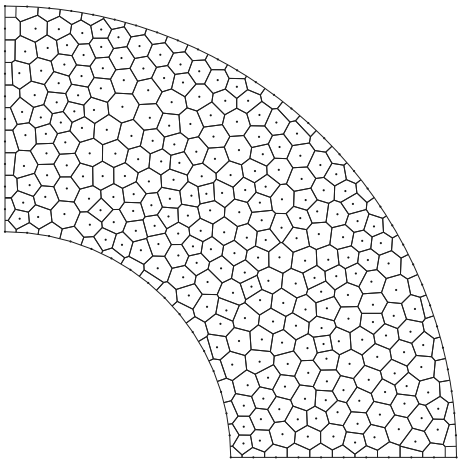

Fig. 4 A clipped Voronoi diagram

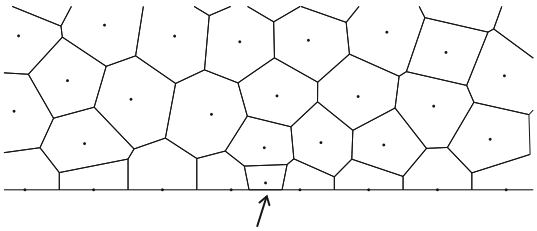

Fig. 5 The arrow shows an interior Voronoi cell that intersects the boundary

3.1.1 Convexity of the clipping polygons

The Sutherland–Hodgman algorithm is a simple algorithm that can *clip both concave and convex polygons against convex clipping polygons* [1], as seen on Fig. 6. It could prove itself useful for clipping the Voronoi cells with the boundaries [26]. However, before applying the algorithm, it is necessary to determine if the clipping polygons made of boundary line segments are locally convex.

The Voronoi cell of an interior mesh vertex can sometimes intersect the boundary, as shown on Fig. 7. The Voronoi cell on the figure is clipped by a single line segment, which can be considered an half-plane. An half-plane is evidently a convex polygon. It is the only possible scenario. It is true that a concave clipping polygon could be built by putting a boundary mesh vertex inside the Voronoi cell. Nevertheless, it would mean that there are more than one vertex inside a single Voronoi cell, which is impossible by definition. The Sutherland–Hodgman algorithm can therefore be applied to interior Voronoi cells.

The Voronoi cell of a boundary mesh vertex always intersects the two boundary line segments at its left and at its right, as seen on Figs. 4 and 5. These particular intersections cannot be treated by the Sutherland–Hodgman algorithm. However, they can be computed by cycling through the Delaunay elements incident to the boundary vertex, as shown on Fig. 8.

A boundary Voronoi cell can also intersect supplementary line segments, such as segment #2 of Fig. 9. Line segment #2 is equivalent to a convex polygon. The Sutherland–Hodgman algorithm can be applied here as well.

3.1.2 Research by propagation

The Voronoi cells of interior vertices can sometimes intersect the boundary. Nevertheless, checking all interior Voronoi cells for intersection with all boundary line segments would take too long. Fortunately, Yan and *al*. discovered an efficient way to perform this task [26].

As seen on Fig. 10, the facet f of the Voronoi cell of mesh vertex $\boldsymbol{x}_1$ intersects the boundary line segment l. By definition, f is the orthogonal bisector of the Delaunay edge joining $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, $\boldsymbol{x}_2$ being another mesh vertex. f is also a facet of $\boldsymbol{x}_2$'s Voronoi cell. Therefore, $\boldsymbol{x}_2$'s Voronoi cell will intersect l. By starting from the Voronoi cells of the boundary mesh vertices, it is possible to identify all interior Voronoi cells that intersect the boundary while only verifying a small amount of them.

3.1.3 Voronoi diagram clipping algorithm

The four steps of the complete clipping algorithm can now be described. These steps are inspired from [26]. Two data structures will be used: a queue and an array of lists. The queue will contain the mesh vertices to be treated. The array will contain for each mesh vertex the list of boundary line segments that intersect its Voronoi cell.

1. The first step consists of creating the interior Voronoi cells, as shown in Fig. 11a. Each Voronoi cell is a polygon composed of Voronoi vertices. The Voronoi vertices are the center of the circles circumscribing the Delaunay triangles. Some interior Voronoi cells might intersect the boundary, but this will be addressed later in steps 3 and 4.
2. There is a boundary line segment at the left and at the right of each boundary mesh vertex. The second step consists of clipping the boundary Voronoi cells with these line segments, as shown in Fig. 11b. For each intersection, the procedure described in Sect. 3.1.2 is used in order to determine if there is a neighboring interior Voronoi cell that intersect the same boundary line segment. If it is the case, then the interior mesh vertex in question is pushed in the queue and the boundary line segment is added to its list.
3. The third step consists of identifying all the boundary line segments that intersect each Voronoi cell with the procedure described in Sect. 3.1.2. Both the queue and

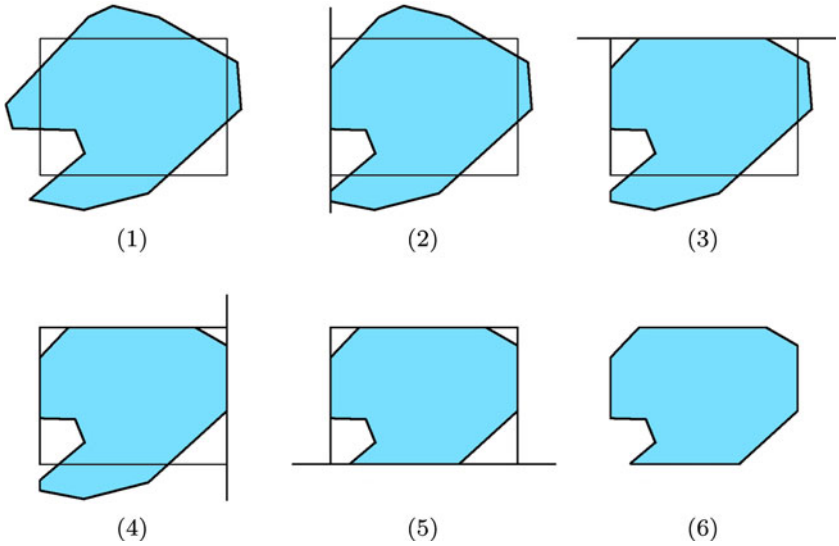

Fig. 6 A six steps demonstration of the Sutherland–Hodgman algorithm

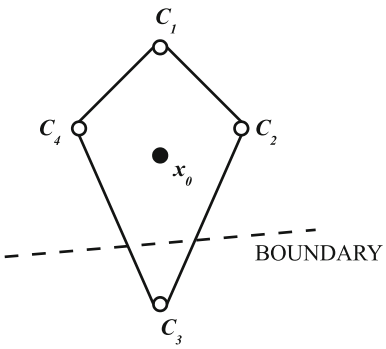

Fig. 7 Interior Voronoi cell intersected by a line segment (throughout this article, the *black dots* are assumed to be mesh vertices and the *hollow dots* are assumed to be Voronoi vertices)

the array of lists are used for this step. The mesh vertices are popped one after the other until the queue is empty. Every time that a mesh vertex is popped, each facet of its Voronoi cell is tested for intersection with each boundary line segment in its list. If there is an intersection, then the other mesh vertex that share the same facet is pushed in the queue, but only if its list did not originally contained the boundary line segment. This precaution is taken in order to ensure that the algorithm does not continue indefinitely.

4. The list of boundary line segments that intersect each Voronoi cell is now known. The last step consists of using the Sutherland–Hodgman algorithm to clip the Voronoi cells with the boundary line segments. Figure 11c shows the final clipped Voronoi diagram.

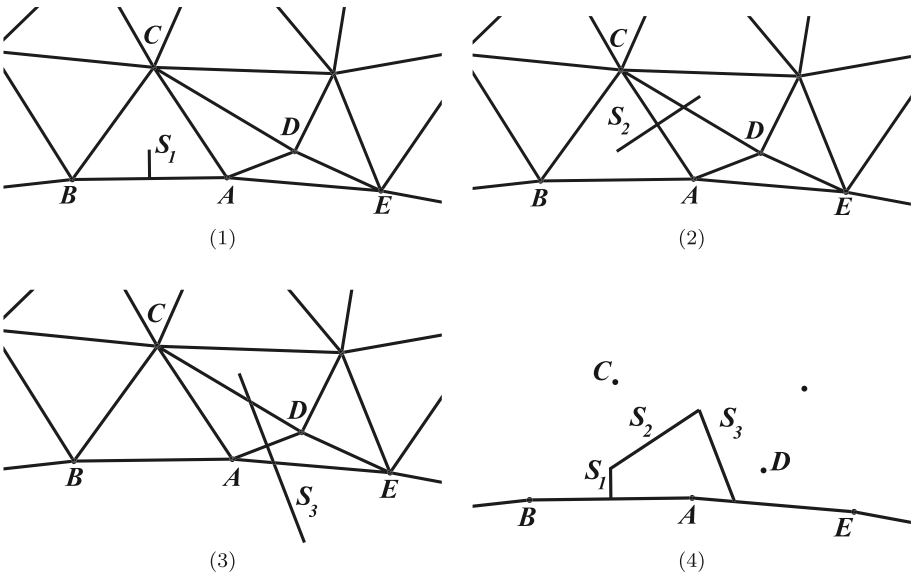

Fig. 8 S_1, S_2 and S_3 are the bisectors of the Delaunay edges *AB, AC* and *AD*

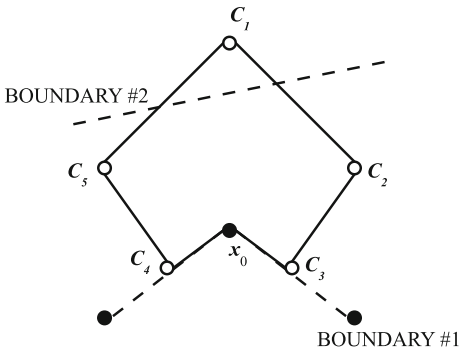

Fig. 9 Boundary Voronoi cell intersected by a line segment

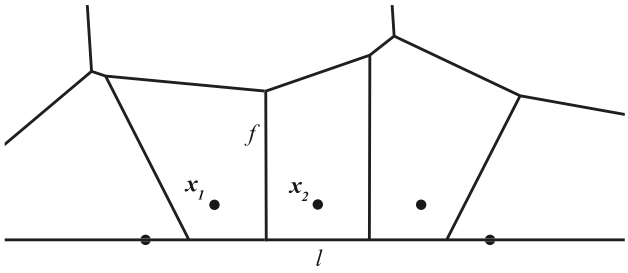

Fig. 10 Three interior Voronoi cells intersecting the boundary

3.2 Energy and gradient evaluation

Sections 3.2.1 and 3.2.2 introduce the energy functional and the gradient. Section 3.2.3 shows how the domain is divided into triangular elements. Section 3.2.4 explains how to compute the energy and the gradient with Gauss integration techniques.

The implementation described in this article can only find local energy minimums. In general, it cannot find global minimums even for simple geometries such as squares and rectangles. It can nevertheless align points in a very satisfactory manner.

For example, in the case of a 11 by 11 square mesh containing 81 interior vertices, the global minimum is readily identifiable. It is a perfect mesh containing only square triangles. Figure 12 illustrates four different initial triangular meshes and their corresponding optimized meshes after 100 iterations. As expected, the global minimum can be found for mesh *A*. Even though meshes *B* and *C* look similar, only mesh *B* can be optimized to the perfect solution. This simple example shows that the resulting mesh depends on the initial configuration. However, the alignment can be improved even in the case of low quality initial meshes.

The variable τ is used to measure how close to perfect square triangles the elements are:

$$\tau = \frac{1}{N}\sum_{i=1}^{N} \max_{j=1,2,3}(\text{abs}(\sin\theta_{ij})) \tag{2}$$

N is the total number of triangular elements in the mesh. θ_{i1} is the first interior angle of triangular element i, θ_{i2} is the second one and θ_{i3} is the third one. τ is equal to 1 for a mesh containing only perfect square triangles. It is equal to 0.866 for a mesh containing only equilateral triangles.

3.2.1 Energy functional

Before defining the energy functional, it is necessary to introduce two parameters: M and ρ.

M specifies the orientation of the quads. It is a function of the orientation angle θ. The orientation angle θ is the angle between the local coordinate system S' and the mesh coordinate system S. The quads will be locally aligned with the axes of S', as shown on Fig. 13.

M is the change of basis matrix that goes from S to S'.

$$M(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

Section 4 describes how to obtain θ in a completely automatic manner for general domains.

The parameter ρ is the density. It specifies the size of the quads. The following formula illustrates the relationship between the density ρ and the mesh size h. It can be obtained with Du procedure [3].

$$\rho \sim \frac{1}{h^{p+2}} \tag{3}$$

The energy functional can now be defined [10] as

$$F(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \sum_{i=1}^{N} \underbrace{\int_{R_i} \rho(\mathbf{y}) ||M(\mathbf{y} - \boldsymbol{x}_i)||_p^p \mathrm{d}\mathbf{y}}_{I^{R_i}(\boldsymbol{x}_i)} \tag{4}$$

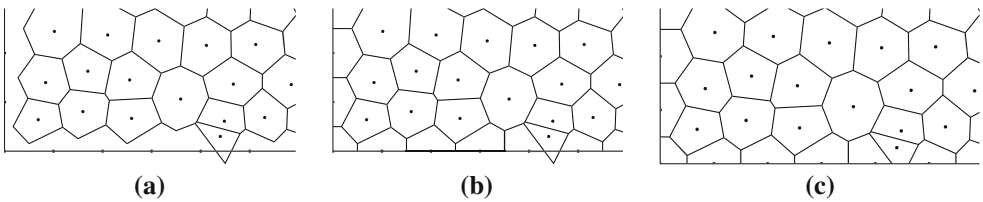

Fig. 11 **a** The unclipped Voronoi cells of the interior mesh vertices, **b** The clipped Voronoi cells of the boundary mesh vertices, **c** The final clipped Voronoi diagram

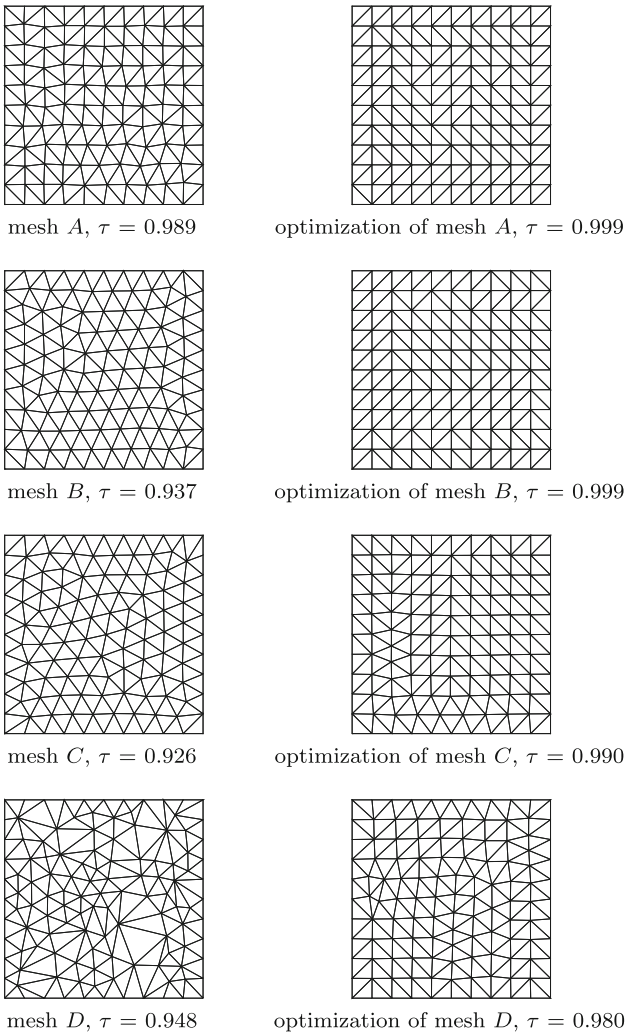

Fig. 12 Meshes *A, B, C* and *D* all contain 81 interior vertices

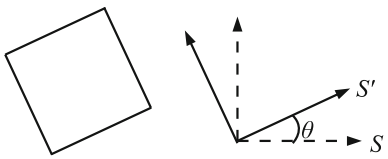

Fig. 13 An oriented quadrilateral element

The index i runs over all N mesh vertices $\boldsymbol{x}_i$. R_i is the domain corresponding to the i-th Voronoi cell generated by mesh vertex $\boldsymbol{x}_i$. In what follows, the energy integral will be denoted by $I^{R_i}(\boldsymbol{x}_i)$. The Voronoi diagram is computed with the L_2 distance, because libraries that can create Voronoi diagrams in other L_p distances are not widely available.

3.2.2 Energy gradient

The energy gradient is the total derivative of F with respect to the position of each non-boundary mesh vertex [10]. Boundary vertices are considered unmovable. It is assumed that $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ are non-boundary vertices and that $\boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+2}, \ldots, \boldsymbol{x}_N$ are boundary vertices. The energy gradient is given by:

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}_k} F(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}_k} \sum_{i=1}^{N} I^{R_i}(\boldsymbol{x}_i) = \sum_{i=1}^{N} \frac{\mathrm{d} I^{R_i}(\boldsymbol{x}_i)}{\mathrm{d}\boldsymbol{x}_k}$$
$$k = 1, \ldots, n$$

The total derivative on the right hand side can be rewritten in terms of partial derivatives.

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}_k} F(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \sum_{i=1}^{N} \frac{\partial I^{R_i}(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_k} + \sum_{i=1}^{N} \frac{\partial I^{R_i}(\boldsymbol{x}_i)}{\partial R_i} \frac{\mathrm{d} R_i}{\mathrm{d}\boldsymbol{x}_k} \tag{5}$$

Equation (5) can be further simplified. First, the partial derivative of $I^{R_i}(\boldsymbol{x}_i)$ with respect to $\boldsymbol{x}_k$ is non-null only when $i = k$. Secondly, R_i can be expanded in terms of the Voronoi vertices $\boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, \ldots, \boldsymbol{C}_{iM_i}$ of the i-th Voronoi cell. All Voronoi cells do not have the same number of vertices M_i. Figure 14 shows a Voronoi cell with six Voronoi vertices. Equation (5) can be rewritten as:

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}_k} F(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \frac{\partial I^{R_k}(\boldsymbol{x}_k)}{\partial \boldsymbol{x}_k} + \sum_{i=1}^{N} \sum_{j=1}^{M_i} \frac{\partial I^{R_i}(\boldsymbol{x}_i)}{\partial \boldsymbol{C}_{ij}} \frac{\mathrm{d}\boldsymbol{C}_{ij}}{\mathrm{d}\boldsymbol{x}_k} \tag{6}$$

Most of the terms $\frac{\mathrm{d}\boldsymbol{C}_{ij}}{\mathrm{d}\boldsymbol{x}_k}$ are null, as explained in the next section.

3.2.3 Gradient assembly

In order to evaluate the integrals with Gauss techniques, the Voronoi cells are divided into triangular elements. Each triangular element is composed of a mesh vertex and two successive Voronoi vertices, as shown in Fig. 14.

Most of the terms inside the double summation found in Eq. (6) vanish. By considering the relationship between the mesh vertices and the Voronoi vertices, the relevant

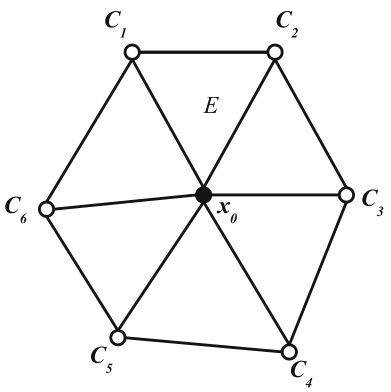

Fig. 14 A Voronoi cell divided into six triangular elements

contributions can be identified. Each Voronoi vertex falls into one of three categories.

1. A Voronoi vertex can be the center of the circle circumscribing a Delaunay element. Voronoi vertex $\boldsymbol{C}_1$ from Fig. 15 belongs to this category. As long as the displacements are infinitesimal, $\boldsymbol{C}_1$ depends only on $\boldsymbol{x}_0, \boldsymbol{x}_1$ and $\boldsymbol{x}_2$. In Figs. 15, 16, 17, the dotted segments are Delaunay edges.
2. A Voronoi vertex can be the intersection point between a Voronoi facet and a boundary line segment. Voronoi vertex $\boldsymbol{C}_2$ from Fig. 16 belongs to this category. Again, as long as the displacements are infinitesimal, $\boldsymbol{C}_2$ depends only on $\boldsymbol{x}_0$, $\boldsymbol{x}_2$ and the boundary line segment.
3. A Voronoi vertex can be the median point between two boundary mesh vertices. Voronoi vertex $\boldsymbol{C}_2$ from Fig. 17 belongs to this category. It depends only on $\boldsymbol{x}_0$ and $\boldsymbol{x}_2$.

The following formulas were derived from Eq. (6). They state the contribution to the energy gradient of the element E found in Fig. 15. It is assumed that $\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x_2}$ and $\boldsymbol{x_3}$ are non-boundary mesh vertices. A+ = sign is used because there will be contributions from other elements as well.

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{x}_0} += \frac{\partial I^E(\boldsymbol{x_0})}{\partial \boldsymbol{x}_0} + \frac{\partial I^E(\boldsymbol{x_0})}{\partial \boldsymbol{C}_1}\frac{\mathrm{d}\boldsymbol{C}_1}{\mathrm{d}\boldsymbol{x}_0} + \frac{\partial I^E(\boldsymbol{x_0})}{\partial \boldsymbol{C}_2}\frac{\mathrm{d}\boldsymbol{C}_2}{\mathrm{d}\boldsymbol{x}_0} \tag{7}$$

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{x}_1} += \frac{\partial I^E(\boldsymbol{x_0})}{\partial \boldsymbol{C}_1}\frac{\mathrm{d}\boldsymbol{C}_1}{\mathrm{d}\boldsymbol{x}_1} \tag{8}$$

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{x}_2} += \frac{\partial I^E(\boldsymbol{x_0})}{\partial \boldsymbol{C}_1}\frac{\mathrm{d}\boldsymbol{C}_1}{\mathrm{d}\boldsymbol{x}_2} + \frac{\partial I^E(\boldsymbol{x_0})}{\partial \boldsymbol{C}_2}\frac{\mathrm{d}\boldsymbol{C}_2}{\mathrm{d}\boldsymbol{x}_2} \tag{9}$$

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{x}_3} += \frac{\partial I^E(\boldsymbol{x_0})}{\partial \boldsymbol{C}_2}\frac{\mathrm{d}\boldsymbol{C}_2}{\mathrm{d}\boldsymbol{x}_3} \tag{10}$$

3.2.4 Gauss integration

The terms inside Eqs. (7) to (10) are integrals on the element E. A linear transformation T will be used in order to go from the reference triangle E' to the triangle E, as shown in Fig. 18. It will then be possible to evaluate the various integrals with Gauss techniques.

$$\boldsymbol{y} = \boldsymbol{T}(u, v) = \boldsymbol{x}_0(1 - u - v) + \boldsymbol{C}_1 u + \boldsymbol{C}_2 v$$

Equation (4) defines $I^E(\boldsymbol{x_0})$ as the energy contribution of the element E. This equation can be rewritten in term of the reference element E', where J is the Jacobian of the transformation T.

$$I^E(\boldsymbol{x_0}) = \int_{E'} \rho(\boldsymbol{T}(u,v))||M(\boldsymbol{T}(u,v) - \boldsymbol{x}_0)||_p^p J \mathrm{d}u\mathrm{d}v \tag{11}$$

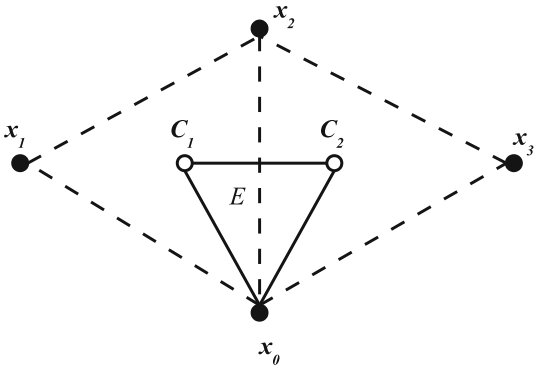

Fig. 15 $\boldsymbol{C}_1$ is the center of the circle circumscribing the Delaunay element $\boldsymbol{x}_0 - \boldsymbol{x}_1 - \boldsymbol{x}_2$

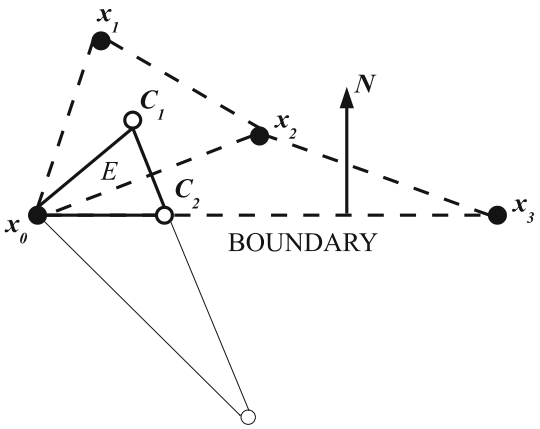

Fig. 16 $\boldsymbol{C}_2$ is the intersection point between the bisector of the Delaunay edge $\boldsymbol{x}_0 - \boldsymbol{x}_2$ and the boundary line segment $\boldsymbol{x}_0 - \boldsymbol{x}_3$

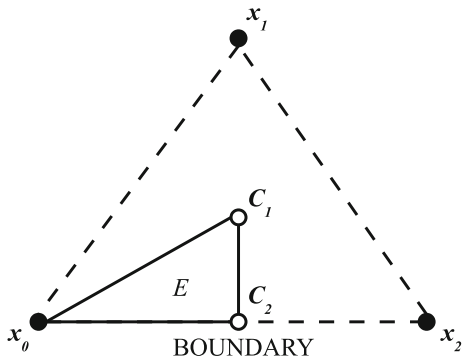

Fig. 17 $\boldsymbol{C}_2$ is the median point between $\boldsymbol{x}_0$ and $\boldsymbol{x}_2$

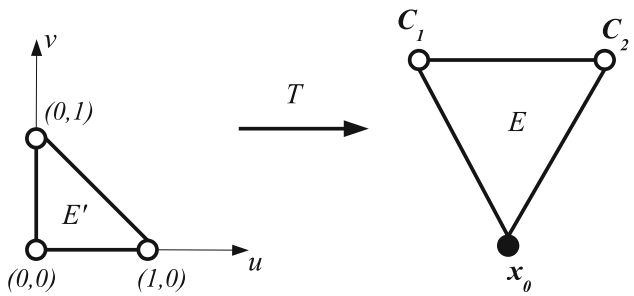

Fig. 18 The linear transformation $\boldsymbol{T}$

Using Eq. (11), the partial derivative of $I^E(\boldsymbol{x_0})$ with respect to the position of mesh vertex $\boldsymbol{x}_0$ can be written as:

$$\frac{\partial I^E(\boldsymbol{x_0})}{\partial \boldsymbol{x}_0} = \frac{\partial}{\partial \boldsymbol{x}_0} \int_{E'} \rho(\boldsymbol{T}(u,v)) ||M(\boldsymbol{T}(u,v) - \boldsymbol{x}_0)||_p^p J \mathrm{d}u\mathrm{d}v$$
$$= \int_{E'} \rho(\boldsymbol{T}(u,v)) \frac{\partial ||M(\boldsymbol{T}(u,v) - \boldsymbol{x}_0)||_p^p}{\partial \boldsymbol{x}_0} J \mathrm{d}u\mathrm{d}v$$

The partial derivative of $I^E(\boldsymbol{x_0})$ with respect to the position of Voronoi vertex $\boldsymbol{C}_1$ is given by:

$$\frac{\partial I^E(\boldsymbol{x_0})}{\partial \boldsymbol{C}_1} = \frac{\partial}{\partial \boldsymbol{C}_1} \int_{E'} \rho(\boldsymbol{T}(u,v)) ||M(\boldsymbol{T}(u,v) - \boldsymbol{x}_0)||_p^p J \mathrm{d}u\mathrm{d}v$$
$$= \int_{E'} \frac{\partial \rho(\boldsymbol{T}(u,v))}{\partial \boldsymbol{T}(u,v)} \frac{\partial \boldsymbol{T}(u,v)}{\partial \boldsymbol{C}_1} ||M(\boldsymbol{T}(u,v) - \boldsymbol{x}_0)||_p^p J$$
$$+ \rho(\boldsymbol{T}(u,v)) \frac{\partial ||M(\boldsymbol{T}(u,v) - \boldsymbol{x}_0)||_p^p}{\partial \boldsymbol{T}(u,v)} \frac{\partial \boldsymbol{T}(u,v)}{\partial \boldsymbol{C}_1} J$$
$$+ \rho(\boldsymbol{T}(u,v)) ||M(\boldsymbol{T}(u,v) - \boldsymbol{x}_0)||_p^p \frac{\partial J}{\partial \boldsymbol{C}_1} \mathrm{d}u\mathrm{d}v$$

The Voronoi cells are divided into triangular elements. The value of the density at each element node is obtained from a given size field. The density is then interpolated linearly. The orientation angle is taken as constant by element.

A Voronoi vertex can sometimes depend on three mesh vertices, as in Fig. 15. The following matrix is the derivative of Voronoi vertex $\boldsymbol{C}_1$ with respect to mesh vertex $\boldsymbol{x}_0$ [10]. The derivatives of $\boldsymbol{C}_1$ with respect to mesh vertices $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ can be obtained by replacing $\boldsymbol{x}_0$ with $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$.

$$\frac{\mathrm{d}\boldsymbol{C}_1}{\mathrm{d}\boldsymbol{x}_0} = \begin{bmatrix} (\boldsymbol{x}_1 - \boldsymbol{x}_0)^T \\ (\boldsymbol{x}_2 - \boldsymbol{x}_0)^T \end{bmatrix}^{-1} \begin{bmatrix} (\boldsymbol{C}_1 - \boldsymbol{x}_0)^T \\ (\boldsymbol{C}_1 - \boldsymbol{x}_0)^T \end{bmatrix}$$

A Voronoi vertex can instead depend on two mesh vertices and one boundary line segment, as in Fig. 16. The following matrix needs to be used in this situation [10]. The derivative of $\boldsymbol{C}_1$ with respect to mesh vertex $\boldsymbol{x}_1$ can be obtained by replacing $\boldsymbol{x}_0$ with $\boldsymbol{x}_1$. $\boldsymbol{N}$ is the normal vector to the boundary line segment. It can be multiplied by any non-zero constant without affecting the value of the derivative.

$$\frac{\mathrm{d}\boldsymbol{C}_1}{\mathrm{d}\boldsymbol{x}_0} = \begin{bmatrix} (\boldsymbol{x}_1 - \boldsymbol{x}_0)^T \\ \boldsymbol{N}^T \end{bmatrix}^{-1} \begin{bmatrix} (\boldsymbol{C}_1 - \boldsymbol{x}_0)^T \\ \boldsymbol{0} \end{bmatrix}$$

The derivation of these matrices can be found in Levy's article [10]. It is important to recall that it is only the derivatives with respect to non-boundary mesh vertices that need to be calculated.

3.3 LBFGS optimization

There are many potential issues with the LBFGS optimization step. In this section, the various problems that can be encountered are discussed. Section 3.3.1 examines the possibility of the LBFGS library moving vertices outside the domain. Section 3.3.2 tells how the gradient can become extremely small because of the higher norm. Section 3.3.3 gives details about the convergence of the algorithm.

3.3.1 The problem of points leaving domain

The LBFGS library performs line searches in order to find new lower energy solutions. There is no guarantees that all solutions are going to be restrained to the domain. A mesh vertex could leave the domain, which would cause the code to fail. To prevent this from happening, it is absolutely necessary to check that all mesh vertices are inside the domain before clipping the Voronoi diagram or computing the energy and the gradient. If there is a single vertex that falls outside the domain, the implementation should not try to perform any calculations. Instead, it should automatically set the energy to a very large value, such as 10^9. This will ensure that the LBFGS library does not select this particular solution. The library will continue the line search and will eventually find a lower energy solution with all mesh vertices inside the domain. An octree data structure can be used for efficient spatial searches.

3.3.2 Gradient normalization

The L_6 distance is used in practice. However, even thought a high exponent is necessary in order to align the vertices in a rectangular way, it can also cause numerical problems. If the density ρ is missing from the formulation, the gradient can reach values as low as 10^{-20}. These values are too close to zero and the optimization library will be unable to find acceptable solutions. The addition of a density to the energy functional solves the problem. As seen in Eq. (3), the density is proportional to $h^{-(p+2)}$ and will tend to normalize the energy and the gradient. Both will be closer to one and the optimization library will work properly.

3.3.3 Convergence of the optimization procedure

The energy will get lower and lower after each iteration. Figure 19 shows the convergence curve of a car hood problem. The curve is very steep at the beginning, but nearly horizontal at the end. It is not necessary to reach complete convergence in order to obtain a good solution. Eighty iterations are usually sufficient. Sometimes, instead of converging, a solution can begin to oscillate. This usually happens only after an important decrease in energy. The result is going to be acceptable even in these conditions.

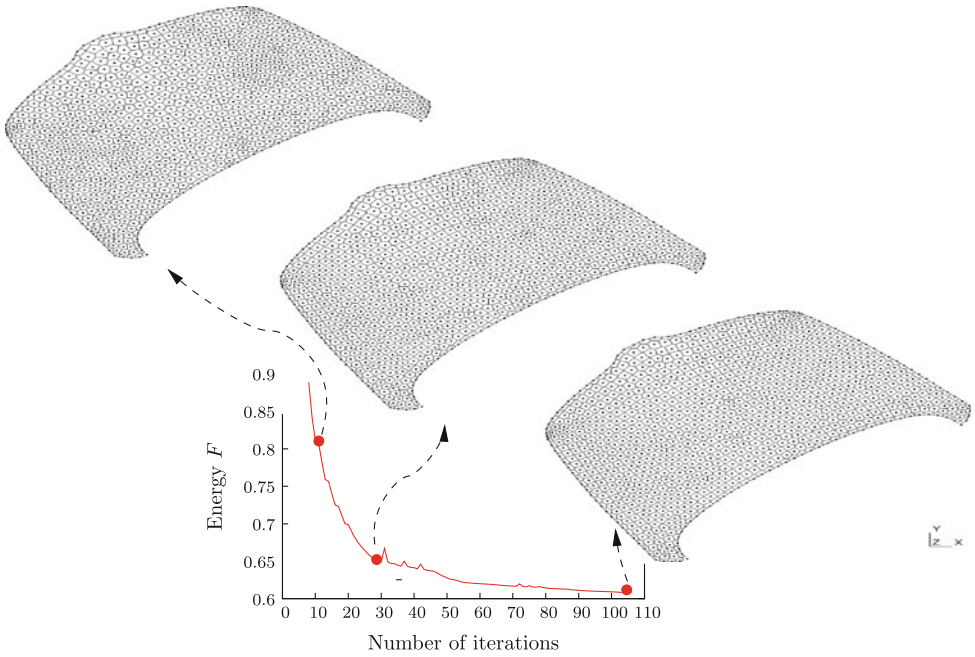

Fig. 19 A convergence curve along with Voronoi diagrams illustrating the process (at each iteration, the clipped Voronoi diagram, the energy and the gradient need to be computed)

4 Computation of the orientation angle

The cross field of Fig. 20 illustrates the alignment direction in a simple domain. A method that could calculate the orientation angle in a completely automatic manner for general geometries would be very useful. If the surface was curved, the local curvature could be used in order to define the orientation [10]. However, it is not the case here. A solution is needed for two-dimensional domains.

As seen in Sect. 3.2.1, the orientation angle θ is the angle between the local coordinate system S' and the mesh coordinate system S. It is a function of the position $\boldsymbol{x}$. In most applications, the quads are expected to be aligned with the boundaries. θ is therefore already known at the boundary. The Laplace equation could prove useful in order to obtain a value of θ inside the domain [20]. Nevertheless, the Laplace equation cannot be directly applied to the angle θ, because all discrete values $\theta + k\frac{\pi}{2}, k \in \mathbb{Z}$ indicate the same orientation. For example, let's consider a square domain aligned with S. At the boundary, the angle θ could be equal to 0, $\frac{\pi}{2}, \frac{3\pi}{2}$, etc. The smoothing process would generate a solution ranging from 0 to a multiple of $\frac{\pi}{2}$, which would lead to an incorrect orientation.

Let z be a complex function of θ.

$$z(\boldsymbol{x}) = a(\boldsymbol{x}) + ib(\boldsymbol{x}) = e^{i4\theta(\boldsymbol{x})} \quad \text{with}$$
$$a = \cos(4\theta(\boldsymbol{x})) \quad \text{and} \quad b = \sin(4\theta(\boldsymbol{x}))$$

z is invariant for all discrete values $\theta + k\frac{\pi}{2}, k \in \mathbb{Z}$.

$$e^{i4(\theta(\boldsymbol{x})+k\frac{\pi}{2})} = e^{i4\theta(\boldsymbol{x})+ik2\pi} = e^{i4\theta(\boldsymbol{x})}$$

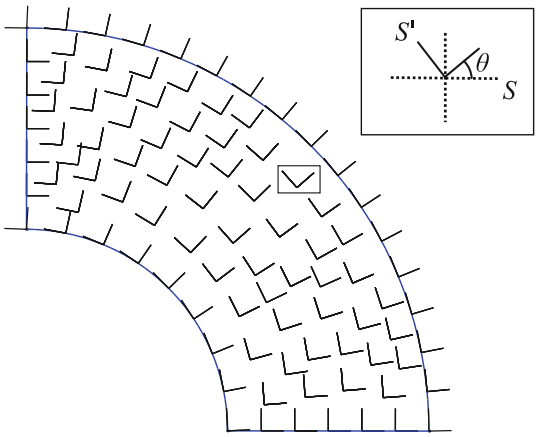

Fig. 20 A smooth cross field illustrating the alignment direction

It is the components of z that are smoothed with the Laplace equation. The values of a and b are already known at the boundary.

$$\nabla^2 a = 0 \quad \text{and} \quad \nabla^2 b = 0 \quad \text{inside the domain}$$
$$a = \cos(4\theta(\boldsymbol{x})) \quad \text{and} \quad b = \sin(4\theta(\boldsymbol{x})) \text{ on the boundaries}$$

Let V be a boundary mesh vertex located at position $\boldsymbol{v}$. V is connected to one boundary edge at its left and one boundary edge at its right. $\theta(\boldsymbol{v})$ is equal to the angle of one of the two boundary edges with respect to the x axis of S.

The modulus of z is equal to one on the boundaries, i.e. $a^2 + b^2 = 1$. However, there is no guarantee that the modulus of z will remain equal to one inside the domain. This is why both values a and b are necessary in order to retrieve the angle θ:

$$\theta(\boldsymbol{x}) = \frac{1}{4}\text{atan2}(b(\boldsymbol{x}), a(\boldsymbol{x}))$$

A similar explanation of this method can be found in [16].

5 Results

In this section, different examples of high quality quad surface meshes with well defined orientations are presented.

Five steps are necessary to produce quad meshes:

1. Compute a conformal mapping that maps the 3D surface to a 2D parametric space [14].
2. Create a triangular mesh in the parametric space.
3. Use this mesh to compute an orientation angle with the method described in Sect. 4.
4. Use Levy's algorithm to optimize the location of the mesh vertices.
5. Apply the Blossom-Quad algorithm described in [17] to combine the triangles into quads. Blossom-Quad uses the well-known Blossom algorithm in order to find a perfect matching between triangles. The matching also optimizes the mean quality of the quads.

Figure 21 illustrates the different steps of the global process: (1) is the triangular mesh of the geometry in the three-dimensional space. (2) is the triangular mesh of the geometry in the parametric space. (3) shows the cross field determining the orientation of the quads. (4) is the triangular mesh in the parametric space after the application of Levy's algorithm. (5) is the final quad mesh.

Figure 22 shows the energy density inside each Voronoi cell after the application of Levy's algorithm. As seen from Eq. (4), the energy density is in fact the value of the L_p distance normalized by the density. The alignment is perfect in pale blue regions. It is more or less incorrect in yellow, orange or red regions.

The L_6 distance was chosen for all the examples of this article. However, it might be useful to compare the results obtained with other distances. Figure 23 shows a car hood mesh optimized with three different distances: L_2, L_4 and L_6.

The variable τ is equal to 0.910 for the L_2 mesh, 0.967 for the L_4 mesh and 0.982 for the L_6 mesh. It has been calculated with Eq. (2). As expected, τ is higher for the L_6 mesh than for the L_4 mesh. The number of Gauss integration points necessary to evaluate the integrals exactly increases with the exponent of the L_p distance. The L_6 distance is therefore a good compromise between speed and squareness.

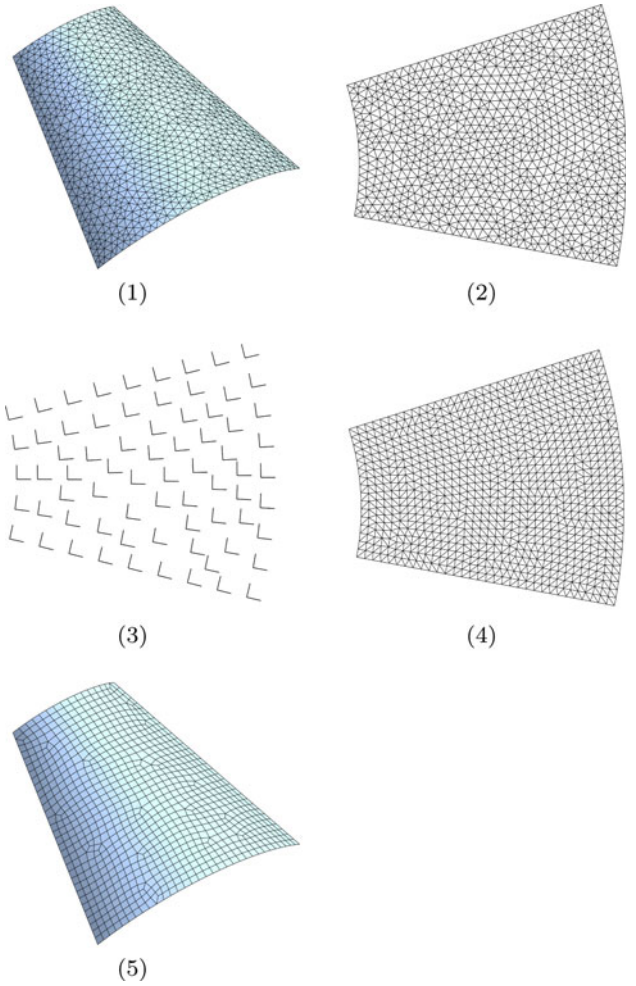

Fig. 21 The various steps necessary to produce quad meshes

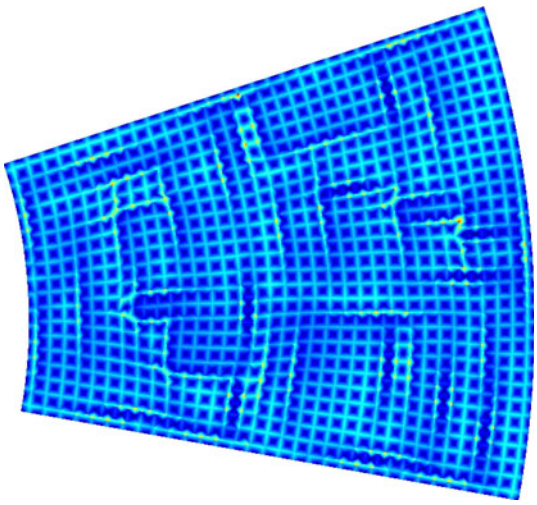

Fig. 22 The energy density in the parametric space after the application of Levy's algorithm

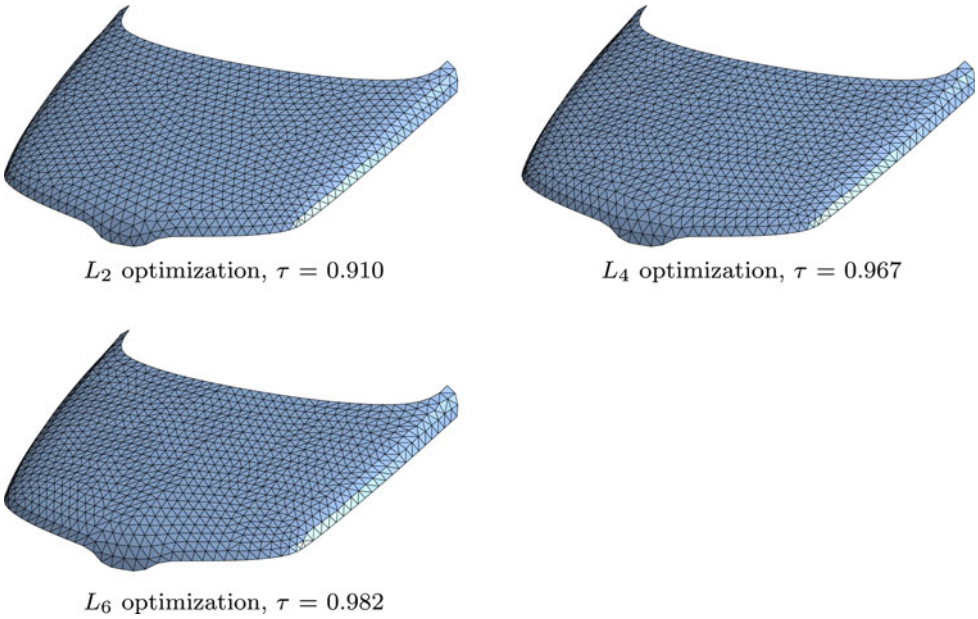

Fig. 23 Three meshes of the same car hood optimized with different L_p distances

In Figs. 24 and 25, the meshes labelled as (b) were created with Levy's algorithm, while the meshes labelled as (a) were created without it. The effect of Levy's algorithm can be verified by comparing (a) and (b).

Figure 24 shows a car body part. (b) contains 10,051 quads. It took 5 min and 58 s to perform the 199 iterations of Levy's algorithm on a standard 2010 laptop. A similar result could have been obtained with approximately 70 iterations in 2 min and 12 s.

(a) has an average quality of $\bar{\eta} = 0.80$ and (b) has an average quality of $\bar{\eta} = 0.90$. The quality of a quadrilateral $\eta(q)$ is defined by the values of its four angles α_k, $k = 1, 2, 3, 4$ [17]:

$$\eta(q) = \max\left(1 - \frac{2}{\pi}\max_k\left(\left|\frac{\pi}{2} - \alpha_k\right|\right), 0\right)$$

The quality goes from zero to one. If the element is a perfect square, it is equal to one.

In an ideal quad mesh, each non-boundary mesh vertex is 4-valent, which means that it is connected to four neighbors. In (a), 71 % of the non-boundary mesh vertices are 4-valent. In (b), this number reaches 89 %.

Figure 25 shows a mechanical part. (b) contains 44,489 quads and 250 geometric faces. It took 17 min and 43 s to perform 199 iterations on each of the faces. (a) has an average quality of $\bar{\eta} = 0.77$, while (b) has an average quality of $\bar{\eta} = 0.84$. (b) contains 962 left-over triangles. These triangles may be eliminated by using an improved version of Blossom-Quad.

A time profiler was used to analyze the performance of the implementation. The data show that calculating the energy and the gradient is very time-consuming. The clipped Voronoi diagram step as well as the LBFGS optimization step have a negligible contribution.

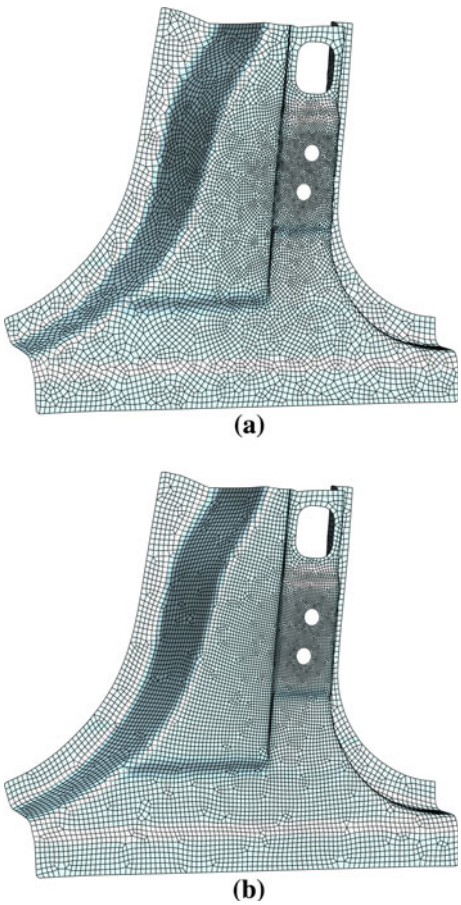

Fig. 24 **a** Blossom-Quad only and **b** Levy and Blossom-Quad

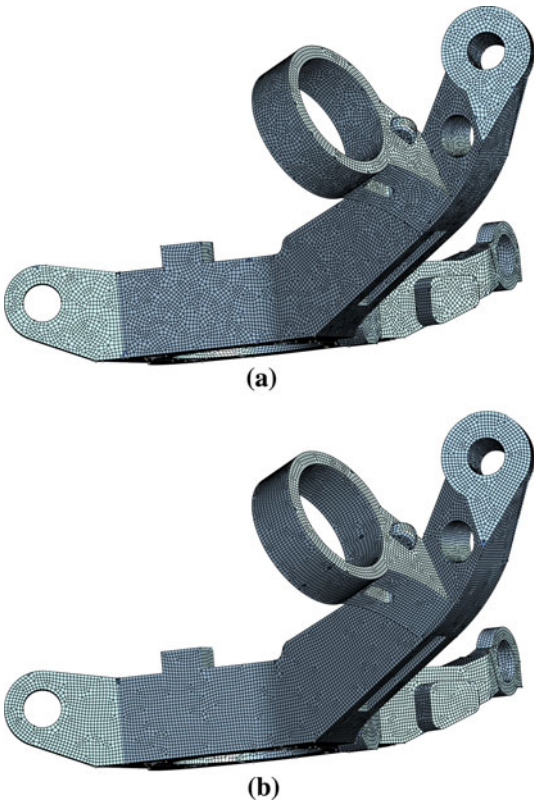

Fig. 25 **a** Blossom-Quad only and **b** Levy and Blossom-Quad

6 Conclusion

A two-dimensional method for the minimization of Lloyd's energy in the L_p norm has been described in this article. It is based on well-known gauss integration techniques and it does not modify the initial edge meshes. When used in combination with an indirect algorithm like Blossom-Quad, it is able to create well-oriented quads of varying size. The orientation field is computed in a completely automatic manner. By taking advantage of global parametrization techniques, three-dimensional surfaces can also be meshed.

The method has two apparent drawbacks. First, the execution time can be sizable. The procedure is much more complex than the traditional Lloyd's algorithm and optimizing large meshes can take very long. Secondly, the method can only find local minimums, not global ones.

Indirect techniques can also be used to create hexahedra. However, in order to create good quality hex meshes aligned in precise directions, a three-dimensional version of Levy's algorithm would be necessary. The implementation described in this article could be used as a starting point. Computing the energy and the gradient would not be particularly more difficult in three dimensions. Nevertheless, clipping a Voronoi diagram in three dimensions would be much more complex, but feasible [26].

Acknowledgments This work has been partially supported by the Belgian Walloon Region under WIST grants ONELAB 1017086 and DOMHEX 1017074.

References

1. Blundell BG (2008) An introduction to computer graphics and creative 3-D environments. Springer, London
2. Bochkanov S, Bystritsky V (1999–2011) Alglib. http://www.alglib.net
3. Du Q, Wang D (2003) Tetrahedral mesh generation and optimization based on centroidal voronoi tessellations. Int J Numer Meth Eng 56:1355–1373
4. Du Q, Faber V, Gunzburger M (1999) Centroidal voronoi tessellations: applications and algorithms. Siam Rev 41:637–676
5. Ferziger J, Peric M (1999) Computational methods for fluid dynamics. Springer, Berlin
6. Hausner A (2001) Simulating decorative mosaics. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques, New York
7. Hughes T (2000) The finite element method: linear static and dynamic finite element analysis. Dover Publications, New York
8. Lai YK, Hu SM, Martin RR (2006) Surface mosaics. Visual Comput 22:604–611
9. Lai YK, Kobbelt L, Hu SM (2010) Feature aligned quad dominant remeshing using iterative local updates. Comput Aided D 42:109–117
10. Levy B, Liu Y (2010) l_p centroidal voronoi tessellation and its applications. In: Hoppe H (ed) ACM Transactions on Graphics, University of California, Los Angeles
11. Liu Y, Wang W, Levy B, Sun F, Yan DM, Lu L, Yang C (2009) On centroidal voronoi tessellation-energy smoothness and fast computation. ACM Trans Graphic 28:1–17
12. Liu Y, Xu W, Wang J, Zhu L, Guo B, Chen F, Wang G (2011) General planar quadrilateral mesh design using conjugate direction field. In: Bala K (ed) ACM Transactions on Graphics, Hong-Kong
13. Liziér MAS, Siqueira MF, Daniels J, Silva CT, Nonato LG (2011) Template-based quadrilateral mesh generation from imaging data. Visual Comput 27:887–903
14. Marchandise E, de Wiart CC, Vos WG, Geuzaine C, Remacle JF (2011) High-quality surface remeshing using harmonic maps-part II: surfaces with high genus and of large aspect ratio. Int J Numer Meth Eng 86:1303–1321
15. Prakash S, Ethier CR (2001) Requirements for mesh resolution in 3d computational hemodynamics. J Biomech Eng 123:134–144
16. Remacle JF, Henrotte F, Baudouin TC, Geuzaine C, Béchet E, Mouton T, Marchandise E (2011) A frontal delaunay quad mesh generator using the l^∞ norm. In: Proceedings of the 20th International Meshing Roundtable, Paris
17. Remacle JF, Lambrechts J, Seny B, Marchandise E, Johnen A, Geuzaine C (2011) Blossom-quad: a non-uniform quadrilateral mesh generator using a minimum cost perfect matching algorithm. Int J Numer Meth Eng 89:1102–1119
18. Roache P (1992) Computational fluid dynamics. Hermosa, Albuquerque
19. Sherwin SJ, Peiró J (2002) Mesh generation in curvilinear domains using high-order elements. Int J Numer Meth Eng 53:207–223
20. Shimada K, Liao JH, Itoh T (1998) Quadrilateral meshing with directionality control through the packing of square cells. In: Proceedings of the 7th International Meshing Roundtable, Dearborn

21. Tarini M, Puppo E, Panozzo D, Pietroni N, Cignoni P (2011) Simple quad domains for field aligned mesh parametrization. In: Bala K (ed) ACM Transactions on Graphics, Hong-Kong
22. Tchon KF, Camarero R (2006) Quad-dominant mesh adaptation using specialized simplicial optimization. In: Proceedings of the 15th International Meshing Roundtable, Birmingham
23. Vinchurkar S, Longest PW (2007) Effect of mesh style and grid convergence on particle deposition in bifurcating airway models with comparisons to experimental data. Med Eng Phys 29:350–366
24. Vinchurkar S, Longest PW (2008) Evaluation of hexahedral, prismatic and hybrid mesh styles for simulating respiratory aerosol dynamics. Comput Fluids 37:317–331
25. Yamakawa S, Shimada K (2003) Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells. Int J Numer Meth Eng 57:2099–2129
26. Yan DM, Wang W, Levy B, Liu Y (2010) Efficient computation of 3d clipped voronoi diagram. In: Mourrain B, Schaefer S, Xu G (eds) GMP 2010 Conference Proceedings, Castro Urdiales
27. Zienkiewicz OC, Taylor RL (2000) The finite element method: the basis, vol 1. Butterworth-Heinemann, Oxford
28. Zienkiewicz OC, Rojek J, Taylor RL, Pastor M (1998) Triangles and tetrahedra in explicit dynamic codes for solids. Int J Numer Meth Eng 43:565–583