Local dual contributions: Representing dual surfaces for block meshing

X. Roca and J. Sarrate*,†

Laboratori de Càlcul Numèric (LaCàN), Departament de Matemàtica Aplicada III, Universitat Politècnica de Catalunya, Jordi Girona 1-3, E-08034 Barcelona, Spain

SUMMARY

In this work we present a tool based on the new concept of local dual contributions for directly representing the topology and the approximated geometry of the dual of a block mesh. This tool allows us to obtain a valid block decomposition of a given geometry without a previous discretization of its boundary. Specifically, our tool is composed of a hierarchical scheme and a set of matching rules that explicitly insert descriptions of dual surfaces and handle their intersections. That is, the proposed tool generates a dual of the block mesh with intersections of the proper multiplicity without gaps and which respects the boundary features of the domain. Finally, we present several examples that illustrate the applicability of the tool.

KEY WORDS: mesh generation; dual; block decomposition; hexahedra; quadrilaterals

1. INTRODUCTION

Hexahedral meshes are still preferred in a wide range of simulations. For instance in applications where a strict alignment of elements is required by the analysis: boundary layers in computational fluid dynamics or composites in structural dynamics. Therefore, the solution of these problems demands a tool that automatically generates hexahedral elements, see [1, 2] for a survey of the available techniques. However, a fully automatic hexahedral mesh generation algorithm for any arbitrary geometry is still not available.

This work has been prompted by the existence of a well-known and straightforward procedure to create unstructured quadrilateral or hexahedral meshes from 2D or 3D triangulations, respectively.

*Correspondence to: J. Sarrate, Laboratori de Càlcul Numèric (LaCàN), Departament de Matemàtica Aplicada III, Universitat Politècnica de Catalunya, Jordi Girona 1-3, E-08034 Barcelona, Spain.

†E-mail: jose.sarrate@upc.edu

Contract/grant sponsor: Spanish *Ministerio de Ciencia e Innovación*; contract/grant numbers: DPI2007-62395, BIA2007-66965, CGL2008-06003-C03-02/CLI

For instance, we can split each mesh triangle into three quadrilaterals by adding a node at the triangle barycenter and a middle node on each edge. Similarly, each mesh tetrahedron can be split into four hexahedra, see [3]. However, meshes obtained with this procedure do not provide enough element quality to use them in a numerical simulation. Therefore the question arises: can we modify a mesh obtained with this procedure in order to obtain a high quality hexahedral mesh?

Inspired by the above question, we consider the dual of a quadrilateral or hexahedral mesh obtained by splitting a coarse initial triangulation. Can we use this initial dual in order to obtain another dual arrangement that leads to an ultra-coarse quadrilateral or hexahedral mesh? To this end, we propose to select part of the entities of the initial dual to obtain a discretized version of the final dual. We select these entities to describe dual surfaces. Therefore, this selection is equivalent to insert layers of hexahedra to the primal. Then we can dualize this discretized dual to obtain a topological decomposition of the domain in *blocks* (ultra-coarse quadrilaterals or hexahedra). Several authors have proposed hexahedral meshing algorithms based on the dual. The *whisker weaving* algorithm proposed by Tautges builds the topology, not the geometry, of the dual of a hexahedral mesh using an advancing front method [4]. Mueller-Hannemann [5] proposed a purely topological method to decompose a prescribed quadrilateral surface mesh into hexahedra. The decomposition is guided by the topology of the closed dual curves of the surface mesh. Calvo *et al.* [6] presented a decomposition approach where closed dual curves are used to recursively divide the topology of the dual of the surface mesh into two topological balls. These algorithms use the combinatorial information contained in the topology of the dual but do not construct its geometric representation.

All the above dual approaches to hexahedral mesh generation start from a prescribed surface quadrilateral mesh that over-constrains the problem. Hence, these methods usually require modification of the prescribed boundary mesh. In order to relax hexahedral meshing problem and avoid these modifications Staten *et al.* recently presented the *unconstrained plastering* approach [7, 8]. This primal approach consists of generating hexahedral elements in an advancing front manner without the constraint (*unconstrained*) of respecting a bounding quadrilateral mesh. Note that, the unconstrained paradigm is implicitly used by the decomposition methods based on the medial axis [9–12]. These methods use the medial axis, the locus of the points which are roughly along the middle of a domain, as a reference to decompose the domain in a coarse hexahedral mesh.

Unconstrained dual surface insertion algorithms, where both the topology and the geometry of the dual are determined, have only been sketched in the literature. In particular, Murdoch *et al.* briefly introduce the *twist plane insertion* algorithm [13], and Calvo speculates about the *free stroking*[‡] method [14]. However these authors did not implement it because a methodology to represent, intersect, and insert *continuous* dual surfaces is not available. On the contrary, here we propose to obtain a *discretized* representation of a block mesh dual by insertion of *discretized* dual surfaces. The components of the discretized dual are selected from the initial dual configuration.

The main contribution of this work is to propose a new tool that allows representing, intersecting, and inserting discretized dual surfaces based on the local dual contributions concept. Moreover, this tool is used to implement an automatic decomposition tool that deals with blocky geometries. The block-meshing procedure is based on the proposed unconstrained dual approach for obtaining topological decompositions in block elements. Specifically, we use a tetrahedral mesh where we

[‡]Named 'libre trazado' in the original work in Spanish.

directly construct the representation of the block mesh dual to finally obtain the primal blocks from it.

The remainder of the paper is organized as follows. First, we present a 2D motivation example in Section 2. According to this motivation, in Section 3 we propose a block meshing approach. To develop this approach we present the required 3D theory in Section 4. Specifically, we introduce the concept of local dual contributions. Then, in Section 5 we detail how to hierarchically add local dual contributions to represent a valid dual of a hexahedral mesh. The main application of these dual representations is the generation of meshes composed by block elements, see Section 6. Finally, in Section 7 we present several examples that show the capabilities of the proposed block meshing approach.

2. 2D MOTIVATION

To illustrate and clarify the proposed block-meshing paradigm we present a 2D example. Specifically, we consider a rectangular domain to be meshed with ultra-coarse quadrilateral elements. A natural requirement is that high quality elements must be placed near the boundary. To this end we require an independent row of quadrilateral elements that follow each boundary curve, see Figure 1(a). The coarsest mesh that fulfills this constraint is presented in Figure 1(b). Connecting each quadrilateral edge with the opposite one, see Figure 1(c), we obtain a set of curves known as the dual curves of the mesh. Figure 1(c) shows a geometric representation of the dual obtained by connecting the midpoint of each quadrilateral edge with the midpoint of the opposite edge. We realize that requiring a row of elements for each boundary curve of a 2D geometry, see Figure 1(a), is equivalent to requiring a dual curve parallel to each boundary curve, see Figure 1(c). Note that the configuration in Figure 1(d) is the desired configuration of dual curves because it leads to the desired quadrilateral mesh in Figure 1(b).

Our goal is to generate an initial dual configuration and use it to obtain the desired one. To create the initial dual configuration we first generate a coarse triangular mesh of the rectangular domain, see Figure 2(a), referred as *reference mesh*. We split each triangle into three quadrilaterals to obtain a *Tri-to-Quad* mesh, see Figure 2(b). To construct the dual curves of this Tri-to-Quad mesh we add for each element two segments of dual curves connecting the midpoints of opposite quadrilateral

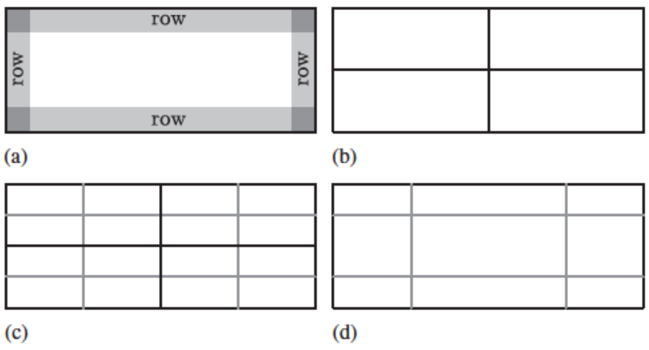

Figure 1. Desired dual curves for a rectangular domain: (a) required rows of quadrilaterals; (b) coarsest quadrilateral mesh; (c) connecting opposite edges; and (d) desired dual curves.

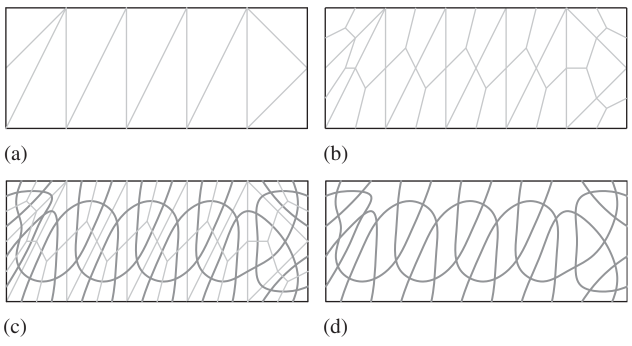

Figure 2. Dual curves for a Tri-to-Quad mesh: (a) reference mesh; (b) Tri-to-Quad mesh; (c) connecting opposite edges; and (d) dual curves.

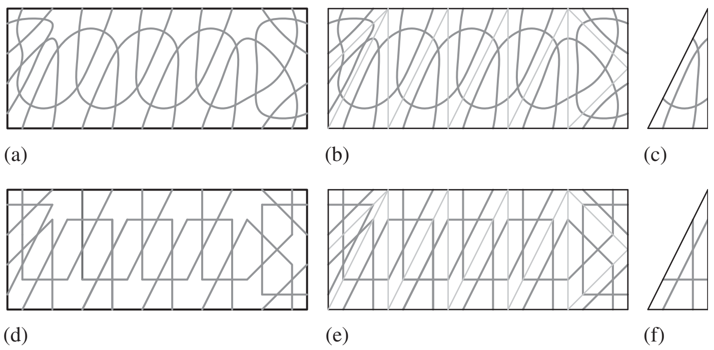

Figure 3. Two topologically equivalent representations of dual curves. Curved representation: (a) curved dual curves; (b) curved dual over the reference mesh; and (c) three curved contributions per triangle. Straight representation with poly-lines: (d) poly-line dual curves; (e) poly-line dual curves over the reference mesh; and (f) three segment contributions per triangle.

edges, see Figure 2(c). These dual edges define a set of dual curves that are in correspondence with rows of quadrilateral elements and define the initial dual configuration, see Figure 2(d).

Consider the initial dual configuration composed of curved dual entities, see Figure 3(a), and add to the foreground the reference mesh triangles, see Figure 3(b). With this representation we conclude that each reference mesh triangle contributes to the dual with three pieces of dual curves, one for each edge, see Figure 3(c). As the dual is a topological representation of the adjacency relations between primal quadrilaterals, we can consider the following representation. Given a reference mesh triangle, we substitute each curved piece of dual curve by a straight segment that will represent a piece of dual curve, see Figure 3(f). Carrying out this substitution on all reference mesh triangles we obtain a dual configuration composed of the straight contributions to the dual of each triangle, see Figure 3(e). Specifically, we obtain a dual configuration composed of poly-line dual curves, see Figure 3(d), that is topologically equivalent to the initial dual representation, see Figure 3(a), and therefore leads to the same quadrilateral mesh.

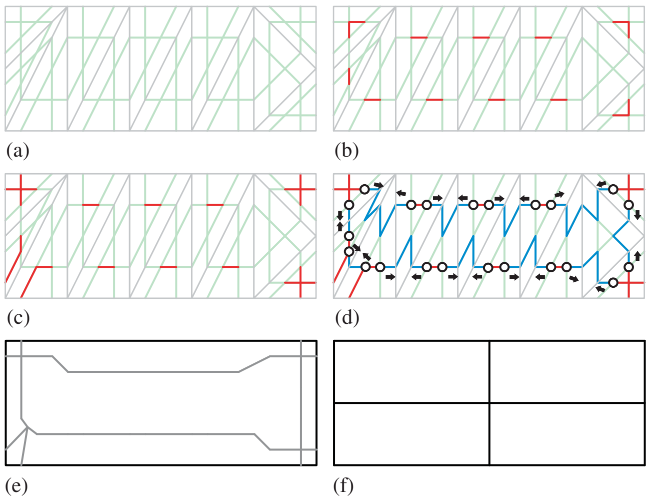
Figure 4. Selecting candidates from the dual of a Tri–Quad mesh: (a) candidate segments; (b) hard contributions that are in front of the boundary and do not intersect the reference mesh; (c) all hard contributions; (d) hard and soft contributions; (e) final dual curves; and (f) final primal mesh.

We have represented with poly-lines the dual curves of the Tri-to-Quad mesh. Now we consider each straight segment of this representation as a potential contributor to the final dual, see Figure 4(a). That is, we want to select several of these *candidate* segments to obtain a final dual configuration. This configuration has to be topologically equivalent to the desired one, represented in Figure 1(d). As a first approach we can select all those candidate segments that are in front of the boundary and do not intersect the reference mesh, see Figure 4(b). Then we add those candidate segments that intersect a reference edge surrounding a vertex feature, see Figure 4(c). These two types of selected candidates will be referred as *hard* contributions. Now we have a set of segments that contribute to the desired dual and lead to an invalid representation with gaps, see Figure 4(c). To fill these gaps we add those candidate segments that are in front of the boundary and intersect the reference mesh, later referred as *soft* contributions, as in Figure 4(d). Note that in our implementation we do not look for gaps or dangling edges while constructing the desired dual. In fact, we follow a set of *matching rules* that ensures that hard and soft contributions are properly added, see Section 5.3. Thus, we have a better representation without gaps. However, we still have a shortcoming. Note that close to the bottom-left corner there are two intersection points, marked with bullets in Figure 4(d), with three incident segments. This configuration is not a valid dual representation because the dual of a quadrilateral mesh only has intersection points with four incident segments. To repair these intersection points we may consider that each group of adjacent segments added at the third step, the soft contributions, are collapsed into one point, see Figure 4. As a result we obtain a new dual configuration, see Figure 4(e), which is topologically equivalent to the desired one, see Figure 1(d). It is important to point out that in our implementation we do not really collapse adjacent soft contributions, see details in Section 4.4.

Finally the block mesh, Figure 4(f), is obtained as the dual of the final dual configuration, Figure 4(e). To this end, we start from the desired dual curves represented by hard and soft contributions, see Figure 5(a). These dual curves determine a set of dual regions, see Figure 5(b). Each

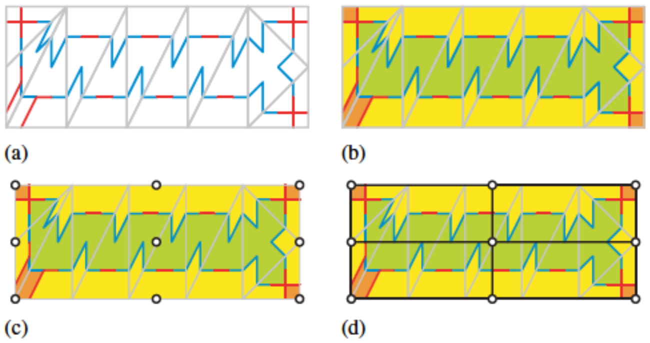

Figure 5. Obtaining the final block mesh: (a) representation of dual curves with hard and soft contributions; (b) dual regions; (c) primal nodes; and (d) primal nodes and edges.

```
Algorithm 1: blockMesh
  Input: geometry to block mesh $\mathcal{G}$
  Output: a block mesh $\mathcal{M}$ of $\mathcal{G}$
1 $\mathcal{R}$ := $obtainReferenceMesh(\mathcal{G})$
2 $\mathcal{D}$ := $addLocalDualContributions(\mathcal{R})$
3 $\mathcal{M}$ := $dualize(\mathcal{D})$
```

dual region is in one-to-one correspondence with a primal node, see Figure 5(c). Moreover, each change of region delimited by at least one hard dual contribution is in one-to-one correspondence with a primal edge, see Figure 5(d). Therefore, we create a primal node inside each dual region and we connect this node with primal nodes in adjacent regions, see details in Section 4.4.

3. ALGORITHM PROPOSAL

We propose a new approach to obtain a valid topological block decomposition of a given domain without a previous discretization of the boundary. The procedure is structured in three steps described in Algorithm 1. In the first step we generate a tetrahedral mesh, see Section 4.2. Then this mesh is used to add planar polygons that define a discrete version of the dual surfaces, see Section 4.3 for definitions and Section 5 for details. This step is equivalent to inserting discrete representations of the dual surfaces. Finally we obtain the block mesh as the dual of the previously generated dual, see Section 4.4.

4. 3D THEORY

In this section we formalize and extend to 3D the concepts presented in Section 2. Furthermore, the definitions follow the order of appearance in the motivation example.

4.1. *Desired local dual*

A valid hexahedral mesh has to fulfill several topological, geometrical, and qualitative requirements to be used in a numerical simulation. A detailed exposition of these conditions for hexahedral meshes can be found in [1, 2]. Our goal is to construct a representation of a hexahedral mesh dual that leads to a valid decomposition in blocks. Therefore, the dual representation has also to meet its own set of topological, geometrical, and qualitative conditions. To this end, we propose to obtain a dual representation that is locally valid, captures the boundary features and fulfills the boundary constraints. Such a representation is said to be the *desired dual*.

4.1.1. Valid local dual. The full set of topological rules to be met by the dual are detailed in [15]. Here we reformulate a part of these rules to state that only four local dual configurations are possible. That is, a dual is *locally valid* if for each point there exists a ball that intersects either:

(i) *no dual entities*, the ball contains a piece of dual region and does not intersect with any dual surfaces, curves or points, see Figure 6(a); or
(ii) *one dual surface*, the ball contains a piece of dual surface and does not intersect with any dual curves or points, see Figure 6(b); or
(iii) *one dual curve*, the ball contains part of a dual curve that is obtained from the intersection of two dual surfaces and does not intersect with any dual point, see Figure 6(c); or
(iv) *one dual point*, the ball contains a dual point that is obtained from the intersection of three dual surfaces, see Figure 6(d).

4.1.2. Boundary features. A hexahedral meshing procedure has to preserve the geometric features near the boundary curves and vertices. To this end, we consider that a geometry curve at the boundary is a *feature curve* if the user or an automatic procedure considers that at least one stack of hexahedra is required around it. To automatically detect the features, we consider the angle θ between the normal vectors of the two adjacent surfaces sharing the curve. According to the notation used for paving [16] and submapping [17] we consider three types of feature curves:

(i) *End*. One stack of hexahedra is required. The user determines this requirement or it is automatically detected if θ is approximately $\pi/2$, see Figure 7(a).
(ii) *Corner*. Three stacks of hexahedra are required. The user determines this requirement or it is automatically detected if θ is approximately $3\pi/2$, see Figure 7(b).
(iii) *Reversal*. Four stacks of hexahedra are required. The user determines this requirement or it is automatically detected if θ is approximately 2π, see Figure 7(c).

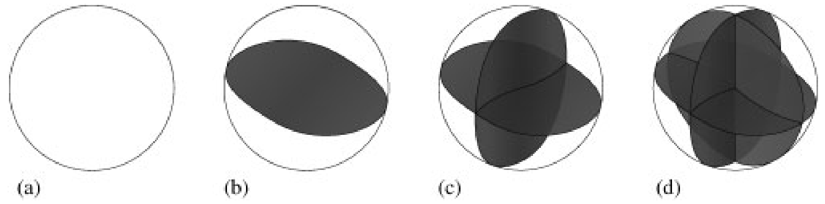

Figure 6. Locally valid dual configurations: (a) no dual surfaces; (b) one dual surface; (c) one dual curve; and (d) one dual point.

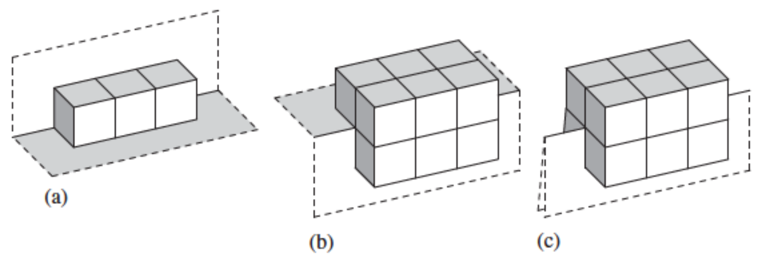

Figure 7. Required meshes around feature curves of type: (a) end, (b) corner, and (c) reversal.

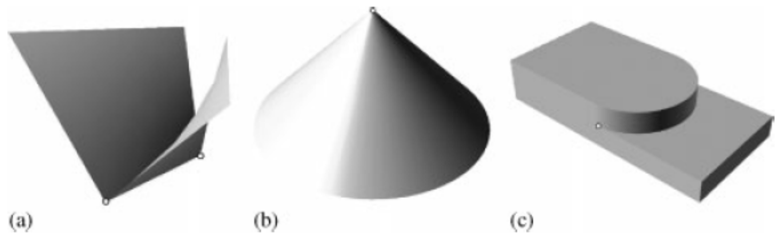

Figure 8. Examples of features not considered in the proposed classification: (a) a curve that begins as corner and ends as reversal; (b) a cone vertex; and (c) a vertex with two adjacent feature curves.

Moreover, we consider that a geometry vertex is a *feature vertex* if it is adjacent to three or more feature curves. Note that the proposed definitions of feature curves and vertices do not take into account features such as those presented in Figure 8. Therefore, in this work we deal with blocky geometries that meet the previous definitions.

4.1.3. Boundary constraints for the dual. Requirements on the geometry and quality of the dual of a hexahedral mesh are less understood than the primal ones. Nevertheless, a detailed survey and analysis of dual constraints associated with hexahedral meshes is presented in [18]. For the purposes of this work we highlight the *boundary constraints*:

(i) For each surface of the domain there exists at least one dual surface, with shape similar to the surface, but offset a certain distance. This ensures at least one layer of primal elements parallel to boundary surfaces.
(ii) For each feature curve of the domain there exists at least one dual curve, with shape similar to the curve, but offset a certain distance. This ensures at least one stack of hexahedra parallel to boundary curves.
(iii) For each feature vertex of the domain there exists at least one dual point offset a certain distance. This ensures at least one hexahedron on the vertices of the domain.

Note that the offset distance is a function of the desired primal element size.

4.2. Reference mesh

To obtain an initial dual configuration we first generate a tetrahedral mesh of the domain referred as *reference mesh*. A tetrahedron of the reference mesh is a *reference element*. Moreover, the

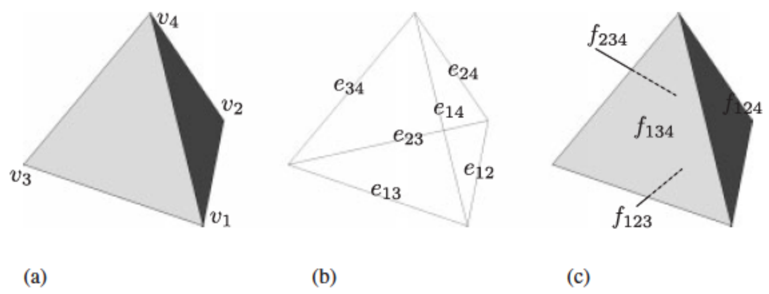

Figure 9. Reference element entities: (a) vertices; (b) edges; and (c) faces.

vertices, edges, and faces of a reference element are denoted by *reference vertices*, *reference edges*, and *reference faces*, respectively. A reference element t with vertex points v_1, v_2, v_3, and v_4 is denoted by $[v_1, v_2, v_3, v_4]$, see Figure 9(a). We univocally denote all reference element edges in t by $e_{ij} := [v_i, v_j]$, where $i<j$ and $i, j = 1, \ldots, 4$. Hence, the six reference edges in t are, see Figure 9(b):

$$e_{12} := [v_1, v_2], \quad e_{13} := [v_1, v_3], \quad e_{14} := [v_1, v_4],$$

$$e_{23} := [v_2, v_3], \quad e_{24} := [v_2, v_4], \quad e_{34} := [v_3, v_4],$$

We univocally denote all reference element faces in t by $f_{ijk} := [v_i, v_j, v_k]$, where $i<j<k$ and $i, j, k = 1, \ldots, 4$. The four reference faces in t are, see Figure 9(c):

$$f_{123} := [v_1, v_2, v_3], \quad f_{124} := [v_1, v_2, v_4], \quad f_{134} := [v_1, v_3, v_4], \quad f_{234} := [v_2, v_3, v_4].$$

4.3. *Local dual contributions*

The basic idea of the proposed method is to generate a valid dual of a decomposition in blocks by selecting pieces, called candidates, of an initial dual configuration. These candidates are planar polygons and determine a planar representation of the initial dual.

4.3.1. Initial dual. From the reference mesh we obtain a hexahedral mesh of the domain by splitting each reference element, see Figure 10(a), in four hexahedra, see Figure 10(b). To this end we use the procedure *hexing the tet* detailed in [3]. The resulting mesh is said to be a *Tet-to-hex mesh*.

Once we obtain the Tet-to-hex mesh, the *initial dual* configuration is obtained by assembling the dual contained in each split reference element, see Figure 11(a). Similar to the 2D case we substitute the part of the dual inside each reference element by four planar surfaces, see Figure 11(b). Both representations, curved and planar, are topologically equivalent. These planes are determined by four plane equations expressed in barycentric coordinates $(\alpha, \beta, \gamma, \delta)$:

$$\alpha = \mu \text{ (plane opposite to vertex } v_1); \quad \beta = \mu \text{ (plane opposite to vertex } v_2);$$

$$\gamma = \mu \text{ (plane opposite to vertex } v_3); \quad \delta = \mu \text{ (plane opposite to vertex } v_4);$$

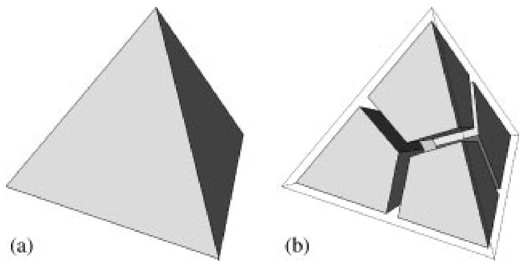

Figure 10. Splitting of a tetrahedron: (a) tetrahedron and (b) decomposition into four hexahedra.

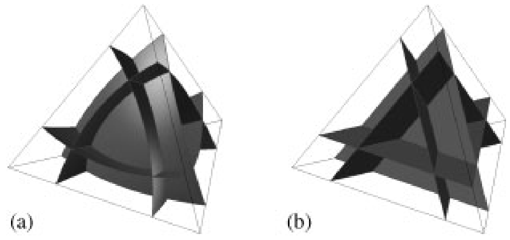

Figure 11. Topologically equivalent pieces of the dual in a reference element: (a) curved representation and (b) planar representation.

where μ is a given number such that $0<\mu<\frac{1}{4}$. Note that we have to add to each plane equation the barycentric coordinates condition $\alpha+\beta+\gamma+\delta=1$, where α, β, γ, $\delta\geqslant 0$. Moreover, we fix the value of μ for the whole reference mesh to ensure that planes match properly between different reference elements. Hence, we obtain a topologically equivalent representation of the curved initial dual with planar surfaces. To obtain the figures in this section we have used $\mu=\frac{1}{5}$. In order to highlight local dual contributions from faces we have used $\mu=\frac{1}{10}$ in Section 7. The selection of these values is due to esthetic criteria and does not have influence on the obtained primal mesh.

4.3.2. Definition and classification of candidates. The assembly of four planar surfaces for each reference element composes our initial dual configuration. Note that the location of these surfaces is determined by a fixed number μ. The planar surfaces intersect themselves inside each reference element. Thus, they can be split into 28 planar polygons. Accordingly, the union of planar polygons composes the initial dual surfaces. A planar polygon inside a reference element is a *candidate* to contribute because it can be selected to compose part of the desired final dual configuration. Note that these 28 planar polygons decompose the reference element into 15 subvolumes.

The candidate polygons inside a reference element $t=[v_1, v_2, v_3, v_4]$ are determined by the intersection points between dual surfaces at reference edges, reference faces, and inside the reference element. For each edge $e_{i,j}=[v_i, v_j]$, where $i<j$ and $i, j=1, \ldots, 4$, we have two intersection points denoted by p_{ij} and p_{ji}. The first index indicates the closest vertex to the intersection point. For instance, the closest vertex to p_{43} is the vertex v_4. In Figure 12(a) we present the two intersection points on edge e_{14}. For each face $[v_i, v_j, v_k]$ we have three intersection points denoted by p_{ijk}, p_{jik}, and p_{kij}, where $i<j<k$ and $i, j, k=1, \ldots, 4$. The first index indicates the closest vertex to the intersection point. Thus, the closest vertex to p_{314} is v_3. In Figure 12(b) we present the three

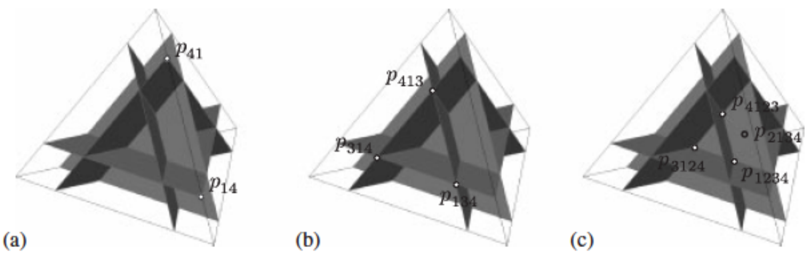

Figure 12. Intersection points of a planar dual representation inside a reference element: (a) points on edge e_{14}; (b) points on face f_{134}; and (c) points inside the reference element.

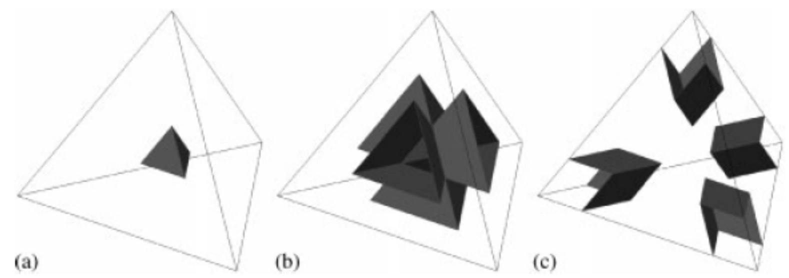

Figure 13. Classification of all possible candidates: (a) face candidates; (b) edge candidates; and (c) vertex candidates.

points on face f_{134}. Finally, inside a reference element t we have four intersection points denoted by p_{1234}, p_{2134}, p_{3124}, and p_{4123}. The first index indicates the closest vertex to the intersection point. For instance, for p_{2134} we have that v_2 is the closest vertex. Figure 12(c) shows all the intersection points inside the reference element. The list of barycentric coordinates for all the intersection points in a reference element is included in Appendix A.1.

We classify the 28 candidate planar polygons of a reference element in three categories. First, the *face candidates* are the candidate polygons that only have intersection points inside the reference element, see Figure 13(a). For each face f in a reference element t we have an associated face candidate denoted by $\mathbf{c}(f;t)$. In Figure 14(a) we present the candidate for face f_{134}. Second, the *edge candidates* are the candidate polygons that touch reference faces but not reference edges, see Figure 13(b). For each adjacent face f to an edge e in a reference element t we have an associated edge candidate denoted by $\mathbf{c}(e;f;t)$. The candidate for edge e_{13} and face f_{134} is presented in Figure 14(b). Finally, the *vertex candidates* are the candidate polygons that touch reference edges, see Figure 13(c). For each adjacent edge e to a vertex v in a reference element t we have an associated vertex candidate denoted by $\mathbf{c}(v;e;t)$. Figure 14(b) shows the candidate for vertex v_3 and edge e_{34}. A list of the 28 candidate polygons for a reference element, with intersection points as vertices, is included in Appendix A.2.

4.3.3. Definition and classification of local dual contributions. We have three types of candidate polygons that can be used to contribute to the final dual configuration. A candidate polygon is considered a *local dual contribution* if it is selected to contribute to the final dual surface.

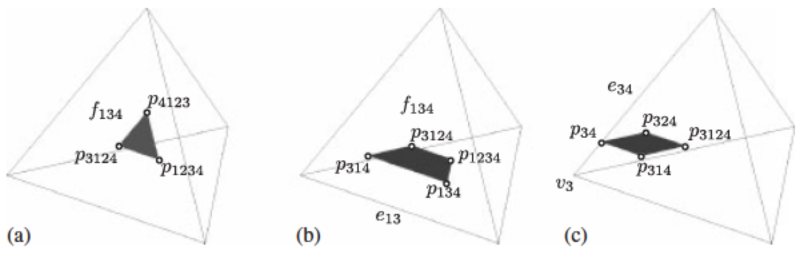

Figure 14. A detailed example for each type of contribution: (a) face candidate for face f_{1234}; (b) edge candidate for edge e_{13} and face f_{134}; and (c) vertex candidate for vertex v_3 and edge e_{34}.

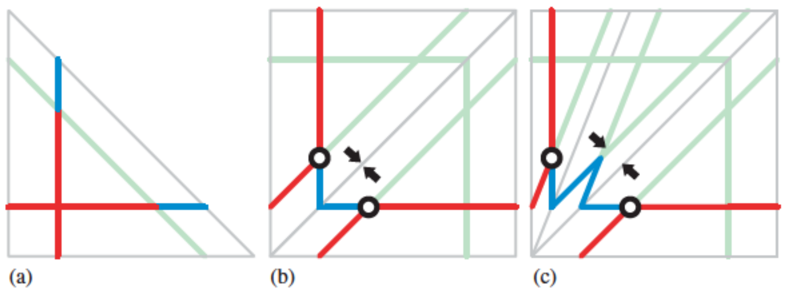

Figure 15. Same local dual configuration for three different reference meshes: (a) one adjacent triangle; (b) two adjacent triangles; and (c) three adjacent triangles.

To represent a valid desired dual we have to ensure that local dual contributions capture the required local features of the desired dual, do not present gaps between them, and match properly. To this end we consider two types of local dual contributions. The first one is denoted by *hard* local dual contribution. This type is used to enforce the required local configurations of the desired dual. Hard local dual contributions are the selected candidates that have to be respected to obtain the desired primal mesh. The second type is called *soft* local dual contribution. It is used to fill gaps and to match properly the local dual contributions between them. Soft local dual contributions have to be interpreted as collapsed to obtain the desired dual, and therefore ignored, to generate the primal mesh, see Section 4.4.

Soft local dual contributions allow obtaining the same local dual configuration for different reference meshes. To illustrate it, consider an end vertex of a 2D domain. Depending on the reference mesh, this end vertex may belong to a different number of reference elements, see Figure 15. Note that, the three presented configurations lead to the same local dual configuration after virtually collapsing the soft local dual contributions.

4.4. *Obtaining the final block mesh*

Once we have a valid representation of the dual we can obtain the block mesh as the dual of this dual. That is, we generate: a primal node for each dual region delimited by dual surfaces; a primal edge connecting primal nodes generated from adjacent dual regions; a primal quadrilateral for each piece of dual curve between two dual points; and a hexahedron for each dual point. However, we have to slightly modify this procedure to take into account hard and soft local dual contributions.

By definition, hard contributions have to be respected and soft contributions have to be interpreted as collapsed into one dual point. According to these definitions, we only generate primal edges between adjacent dual regions separated by at least one local dual contribution of type hard. On the contrary, we do not generate a primal edge between adjacent dual regions that are separated only by local dual contribution of type soft. The last is equivalent to collapsing each soft local dual contribution in one point. Then we can create the quadrilateral faces and finally the hexahedral elements that determine the blocks.

5. ADDING LOCAL DUAL CONTRIBUTIONS

This section details how to add local dual contributions on a reference mesh to obtain the desired dual configuration. To this end, we propose a hierarchical scheme that takes into account a set of matching rules. These rules ensure that adjacent local dual contributions match properly and define locally valid dual configurations. Moreover, these rules depend on three functions that classify the reference entities. These functions enforce that the boundary constraints are fulfilled.

The call graph for the addition of local dual contributions is presented in Table I. Algorithms on the left call algorithms on their right. This is also the order of presentation for the algorithms along this section. Hence, we present the process of adding local dual contributions going from less detail to more detail.

5.1. Hierarchical scheme

We propose a hierarchical scheme to simplify the task of obtaining a valid dual by addition of local dual contributions. This scheme is detailed in Algorithm 2 and it is performed in three steps. First, we add local dual contributions from faces. Second, we add local dual contributions from edges taking into account the face candidates previously added. Third, we add local dual contributions from vertices taking into account the edge candidates added in the previous step. Figure 16 depicts these three steps for an L-shaped geometry.

addLocalDualContributionsFromReferenceFaces. Algorithm 3 describes how to add local dual contributions from faces. Specifically, for each face f inside a reference element t the procedure

Algorithm 2: *addLocalDualContributions*

```
Input: reference mesh $\mathscr{R}$
Output: dual represented by local dual contributions $\mathscr{C}_f$, $\mathscr{C}_e$, and $\mathscr{C}_v$
1 $\mathscr{C}_f := addLocalDualContributionsFromReferenceFaces(\mathscr{R})$
2 $\mathscr{C}_e := addLocalDualContributionsFromReferenceEdges(\mathscr{R}, \mathscr{C}_f)$
3 $\mathscr{C}_v := addLocalDualContributionsFromReferenceVertices(\mathscr{R}, \mathscr{C}_e)$
```

Table I. Call graph for adding local dual contributions.

Depth 1		Depth 2		Depth 3		Depth 4
Algorithm 2	→	Algorithm 3	→	Algorithm 6	→	Algorithm 9
	→	Algorithm 4	→	Algorithm 7	→	Algorithm 10
	→	Algorithm 5	→	Algorithm 8	→	Algorithm 11

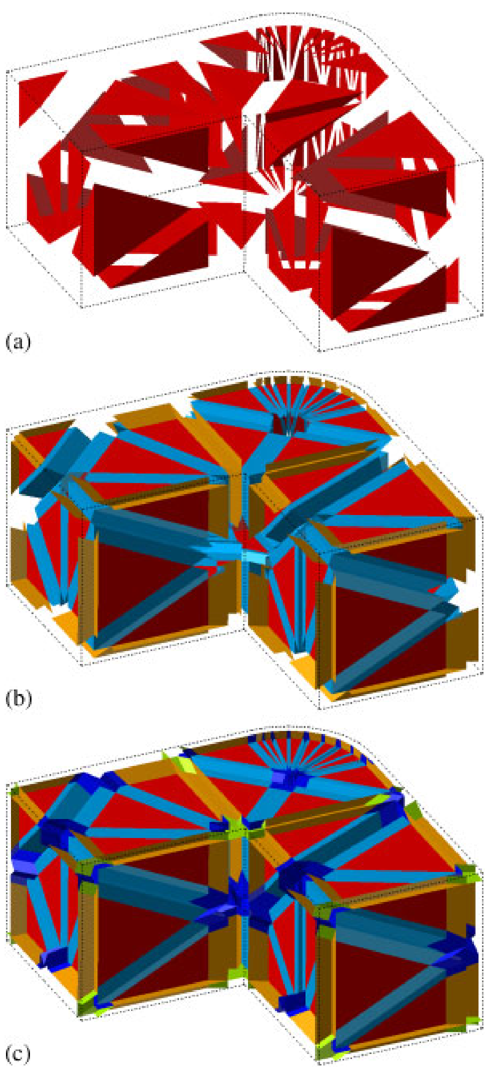

Figure 16. Hierarchical scheme for adding local dual contributions from: (a) faces (red triangles); (b) edges (cyan and orange long quadrilaterals); and (c) vertices (blue and yellow small quadrilaterals). Colors are available in online version of this paper.

checks if the face candidate $\mathbf{c}(f;t)$ has to be added. This checking is performed by the matching rule for faces described later in Algorithm 6. Figure 16(a) shows that several face candidates (triangles) are added as hard local dual contributions (red in online version). These hard local dual contributions are determining pieces of the desired dual surfaces. Note that there are gaps between these contributions.

addLocalDualContributionsFromReferenceEdges. Algorithm 4 takes into account face candidates previously added to add local dual contributions from edges. That is, given an edge e and an adjacent face f the procedure decides if the edge candidate $\mathbf{c}(e;\, f;t)$ has to be considered as

Algorithm 3: *addLocalDualContributionsFromReferenceFaces*

```
Input: reference mesh $\mathscr{R}$
Output: local dual contributions from faces $\mathscr{C}_f$
1 foreach reference element $t$ do
2     foreach reference face $f$ in $t$ do
3         $type := matchingRuleForFace(f, t)$
4         if $type \neq none$ then add $\mathbf{c}(f; t)$ as $type$ in $\mathscr{C}_f$
```

Algorithm 4: *addLocalDualContributionsFromReferenceEdges*

```
Input: reference mesh $\mathscr{R}$
Input: local dual contributions from faces $\mathscr{C}_f$
Output: local dual contributions from edges $\mathscr{C}_e$
1 foreach reference element $t$ do
2     foreach reference edge $e$ in $t$ do
3         foreach adjacent reference face $f$ to $e$ in $t$ do
4             $type := matchingRuleForEdge(e, f, t, \mathscr{C}_f)$
5             if $type \neq none$ then add $\mathbf{c}(e; f; t)$ as $type$ in $\mathscr{C}_e$
```

Algorithm 5: *addLocalDualContributionsFromVertices*

```
Input: reference mesh $\mathscr{R}$
Input: local dual contributions from edges $\mathscr{C}_e$
Output: local dual contributions from vertices $\mathscr{C}_v$
1 foreach reference element $t$ do
2     foreach reference vertex $v$ in $t$ do
3         foreach adjacent reference edge $e$ to $v$ in $t$ do
4             $type := matchingRuleForVertex(v, e, t, \mathscr{C}_e)$
5             if $type \neq none$ then add $\mathbf{c}(v; e; t)$ as $type$ in $\mathscr{C}_v$
```

part of the final dual. This decision is performed by the matching rule for edges, see Algorithm 7 presented later. For instance, Figure 16(b) shows that several edge candidates (long quadrilaterals) are added as soft local dual contributions (cyan in online version) to fill the gaps between the added face candidates. Moreover, several edge candidates are added as hard local dual contributions (orange in online version). These hard contributions determine locally the dual configuration for obtaining the dual curves. Again we still have gaps between added local dual contributions.

addLocalDualContributionsFromVertices. Taking into account the previously added local dual contributions from edges, Algorithm 5 adds the required vertex candidates as local dual contributions. That is, given a vertex v and an adjacent face e we have to decide if the vertex candidate $\mathbf{c}(v; e; t)$ has to be considered as part of the final dual. To this end we use the matching rule for vertices detailed in Algorithm 8. Figure 16(c) shows that vertex candidates (small quadrilaterals) are added as soft local dual contributions (blue in online version) to fill the gaps between the added face and edge candidates. Furthermore, several vertex candidates are added as hard local dual contributions (yellow in online version). These hard contributions determine the proper local dual configuration for representing a dual point. At the end of this step there are no gaps between local dual contributions.

5.2. *Classification of reference entities*

To state the process of adding local dual contributions we have to classify all reference entities, both boundary and inner reference entities. We classify reference entities according to the local dual configuration to be captured around them. Specifically, using the functions presented in Section 5.4 a reference entity is classified as:

(i) *Free*, if dual surfaces, dual curves, and dual points are not required around the reference entity. This type applies to reference faces, edges, and vertices.
(ii) *Layer*, if only dual surfaces are required around the reference entity. Note that dual curves and dual points are not required. This type applies to reference faces, edges, and vertices. To respect the first boundary constraint presented in Section 4.1.3, all boundary faces have to be considered of this type.
(iii) *Stack*, if only dual curves are required around the reference entity. Note that dual points are not required. This type only applies to reference edges and vertices. All the feature edges require at least one stack of hexahedra. Therefore, they are also of type stack. This ensures the second boundary constraint presented in Section 4.1.3.
(iv) *Hexahedron*, if at least one dual point is required around the reference entity. This type applies only to reference vertices. All the feature vertices require at least one hexahedron. Thus, they are also of type hexahedron. This ensures the third boundary constraint presented in Section 4.1.3.

Note that this classification does not take into account the number of dual entities required around a reference entity. In Section 5.4 we detail our implementation for the classification of reference: faces (Algorithm 9), edges (Algorithm 10), and vertices (Algorithm 11).

5.3. *Matching rules*

To ensure that we obtain the desired dual configuration we consider a set of matching rules. On the one hand, matching rules ensure that added candidates match properly and represent a dual without gaps. To this end, local dual contributions of type soft are added. On the other hand, matching rules enforce the desired dual configuration by adding hard local dual contributions.

As the addition process is performed in a hierarchical manner there are three matching rules. That is, there are matching rules for adding local dual contributions from reference faces, edges, and vertices. These rules are based on the classification of the reference entities and the previously added local dual contributions.

matchingRuleForFace. Given a reference face f and its classification we decide to add local dual contributions from this face according to the rule described in Algorithm 6.

Algorithm 6: *matchingRuleForFace*

```
Input: reference face $f$, element $t$
Output: type of local dual contribution to add, type
1 switch classifyFace($f$) do
2     case free
3         type:=none
4     case layer
5         type:=hard
```

Algorithm 7: *matchingRuleForEdge*

```
Input: reference edge $e$, face $f$, and element $t$
Input: local dual contributions from faces $\mathscr{C}_f$
Output: type of local dual contribution to add, type
1 switch classifyEdge($e$) do
2     case free
3         type:=none
4     case layer
5         if $\mathbf{c}(f;t)$ is not in $\mathscr{C}_f$ then type:=soft else type:=none
6     case stack
7         if $\mathbf{c}(f;t)$ is not in $\mathscr{C}_f$ then type:=soft else type:=hard
```

matchingRuleForEdge. Given an edge e and an adjacent face f we have to decide if the edge candidate $\mathbf{c}(e; f; t)$ has to be considered as part of the final dual. Moreover, we have to ensure that the addition of the candidate $\mathbf{c}(e; f; t)$ matches properly with the previously added local dual contributions from faces. To this end we consider a matching rule determined by the classification of edge e and the type of local dual contribution added from face f, see Algorithm 7.

matchingRuleForVertex. Given a vertex v and an adjacent face e we have to decide if the vertex candidate $\mathbf{c}(v; e; t)$ has to be considered as part of the final dual. Moreover, we have to ensure that the addition of the candidate $\mathbf{c}(v; e; t)$ matches properly with the previously added local dual contributions from faces. Algorithm 8 specifies this matching rule according to the classification of vertex v and the type of local dual contributions added from adjacent edge e.

5.4. *Classification functions for reference entities*

According to the proposed hierarchical scheme, the classification functions for reference entities are also implemented in a hierarchical manner. Specifically, the classification function for vertices depends on the classification function for edges. Similarly, the classification of edges depends on the classification function for faces. The latter depends on an initial list $\mathscr{L}$ of reference faces. This list contains the inner reference faces to be considered of type layer. That is, these layer faces determine the inner cuts of the decomposition process, see details in Section 6.1.

In order to illustrate how list $\mathscr{L}$ is filled we consider a 2D analogy. Figure 17(a) presents the reference mesh of an L-shaped planar geometry. To fill the list $\mathscr{L}$ we mark all boundary sides (faces in 3D) and those inner sides (faces in 3D) that have similar normal vectors to the boundary sides adjacent to the corner vertex, see Figure 17(b). Then, we add all the candidate segments that are in front of a marked side (face in 3D) and do not intersect the reference mesh, see Figure 17(c). Note that the inner sides determine two hard contributions. These two hard contributions delimit part of an inner cut. Finally, the remaining contributions are added according to the proposed hierarchical scheme, see Figure 17(d).

classifyFace. The implementation details for classifying reference faces are presented in Algorithm 9. Note that this function classifies as layer those faces being on a geometry surface. That is, the reference faces at the boundary determine that at least one layer of hexahedra has to be generated parallel to boundary surfaces. Therefore, this function enforces the first boundary constraint. The rest of the dual surfaces are determined by the election of the initial list $\mathscr{L}$. Moreover, this list determines the classification of reference edges and vertices.

```
Algorithm 8: matchingRuleForVertex
  Input: reference vertex $v$, edge $e$, and element $t$
  Input: local dual contributions from edges $\mathscr{C}_e$
  Output: type of local dual contribution to add, type
1 $\{f_1, f_2\}$:=adjacentFaces($e$)
2 $\mathbf{c}_1$:=$\mathbf{c}(e; f_1; t)$
3 $\mathbf{c}_2$:=$\mathbf{c}(e; f_2; t)$
4 switch classifyVertex($v$) do
5     case free
6         type:=none
7     case layer
8         if $\mathbf{c}_1$ is not in $\mathscr{C}_e$ and $\mathbf{c}_2$ is not in $\mathscr{C}_e$ then
9             type:=soft
10        else
11            type:=none
12    case stack
13        if $\mathbf{c}_1$ is not in $\mathscr{C}_e$ and $\mathbf{c}_2$ is not in $\mathscr{C}_e$ then
14            type:=soft
15        else if neither $\mathbf{c}_1$ is hard nor $\mathbf{c}_2$ is hard then
16            type:=soft
17        else
18            type:=none
19    case hexahedron
20        if $\mathbf{c}_1$ is not in $\mathscr{C}_e$ and $\mathbf{c}_2$ is not in $\mathscr{C}_e$ then
21            type:=soft
22        else if neither $\mathbf{c}_1$ is hard nor $\mathbf{c}_2$ is hard then
23            type:=soft
24        else
25            type:=hard
```

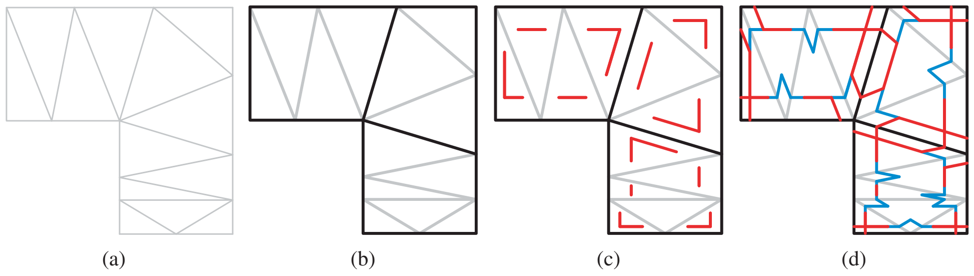
Figure 17. Initial list of reference sides: (a) reference mesh; (b) marked sides; (c) hard dual contributions associated to marked sides; and (d) hard and soft contributions.

classifyEdge. Algorithm 10 presents the implementation for classifying reference edges. This function classifies as stack those edges that are on a feature curve. These stack edges determine that at least one stack of hexahedra has to be generated parallel to the boundary curves. Hence, this

Algorithm 9: *classifyFace*

```
  Input: reference face $f$
  Input: a list of inner reference faces to be considered as layer faces $\mathscr{L}$
  Output: classification of $f$, classification
1 if $f$ is on a geometry surface or in $\mathscr{L}$ then
2     classification:=layer
3 else
4     classification:=free
```

Algorithm 10: *classifyEdge*

```
   Input: reference edge $e$
   Output: classification of $e$, classification
1  if $e$ is on a feature curve then
2      classification:=stack
3  else
4      nOfAdjacentLayerFaces:=number of adjacent faces of type layer
5      switch do
6          case $nOfAdjacentLayerFaces = 0$
7              classification:=free
8          case $0 < nOfAdjacentLayerFaces < 3$
9              classification:=layer
10         case $3 \leqslant nOfAdjacentLayerFaces$
11             classification:=stack
```

function ensures the second boundary constraint. The rest of the classification is done according to the number of adjacent faces of type layer, see line 4. Note that to obtain this number we call the classification function for faces.

classifyVertex. The implementation of the classification function for vertices is detailed in Algorithm 11. This implementation classifies reference vertices coincident with geometry vertices as being of hexahedron type. That is, these vertices are determining that at least one hexahedron has to be generated close to the boundary vertices. Therefore, this function ensures the third boundary constraint. The rest of the classification is done according to the number of adjacent edges of type layer and stack, see lines 4 and 5, respectively. Note that to obtain these numbers we have to call the classification function for edges.

6. APPLICATION TO BLOCK DECOMPOSITION

The addition of local dual contributions is performed according to the classification functions for faces, edges, and vertices. These functions depend on an initial list $\mathscr{L}$ of inner reference faces. This list determines the inner reference faces that are considered to be of layer type. Hence, to automatically generate block meshes with the proposed algorithm, see Section 3, we have to automatically select the layer faces. To illustrate this, Algorithm 12 details a simple procedure to

Algorithm 11: *classifyVertex*

```
Input: reference vertex $v$
Output: classification of $v$, classification
1  if $v$ is on a feature vertex then
2      classification:=hexahedron
3  else
4      nOfAdjacentLayerEdges:=number of adjacent edges of type layer
5      nOfAdjacentStackEdges:=number of adjacent edges of type stack
6      switch do
7          case $nOfAdjacentStackEdges = 0$
8              if $nOfAdjacentLayerEdges = 0$ then
9                  classification:=free
10             else
11                 classification:=layer
12         case $0 < nOfAdjacentStackEdges < 3$
13             classification:=stack
14         case $3 \leqslant nOfAdjacentStackEdges$
15             classification:=hexahedron
```

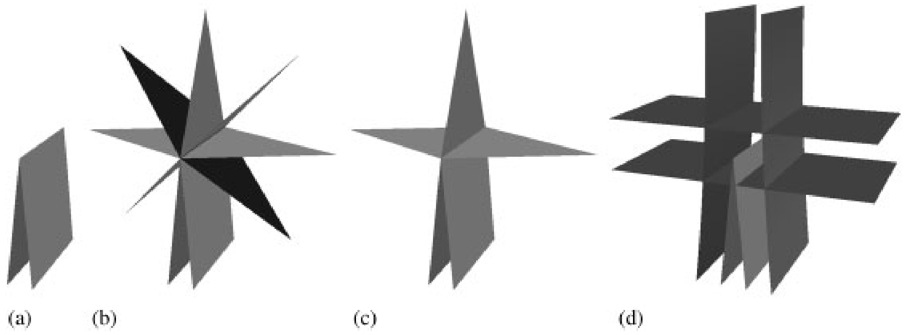

Figure 18. Marking faces around a reversal curve: (a) boundary surfaces of the reversal curve; (b) inner faces adjacent to reversal curve; (c) marked faces; and (d) associated local dual configuration.

mark reference faces as layers based on geometrical criteria. We detail an algorithm that does not deal with reversal curves because the logic is simpler. However, Algorithm 12 can be extended to decompose blocky geometries with reversal curves, see Figure 18(a). To this end, we would have to consider a procedure that marks three adjacent inner faces, Figure 18(b), for each reversal reference edge, see Figure 18(c). Each one of these inner faces corresponds to two parallel pieces of the final dual, see Figure 18(d).

It is important to point out that Algorithm 12 is robust from the topological point of view. That is, the obtained list $\mathscr{L}$ will always lead to a topologically valid dual. However, the quality of the corresponding block mesh could be improved by changing the geometrical predicate of line 15 of Algorithm 12.

Algorithm 12: *markLayers*

```
Input: reference mesh $\mathscr{R}$
Output: faces marked as layers $\mathscr{L}$
1  $\mathscr{L} := \emptyset$
2  $\mathscr{E}_c := \emptyset$
3  $\mathscr{E}_n := \emptyset$
4  foreach feature edge e in $\mathscr{R}$ do
5      if e is on a corner or touches a vertex with valence $>3$ then
6          add to $\mathscr{E}_c[e]$ the two boundary faces adjacent to e
7  oldNOfLayers $:= 0$
8  newNOfLayers $:= 1$
9  while oldNOfLayers$<$newNOfLayers do
10     oldNOfLayers := size of $\mathscr{L}$
11     foreach edge e indexed in $\mathscr{E}_c$ do
12         foreach face f in list $\mathscr{E}_c[e]$ do
13             candidates:=list of inner faces adjacent to e and different to f
14             $\mathbf{n}$:=direction of normal vector to f
15             layer:=select in candidates a face with a normal direction similar to $\mathbf{n}$
16             add to $\mathscr{L}$ face layer
17             foreach edge $e_f \neq e$ in face layer do
18                 if $e_f$ is inner edge then add layer to $\mathscr{E}_n[e_f]$
19     newNOfLayers := size of $\mathscr{L}$
20     $\mathscr{E}_c := \mathscr{E}_n$
21     clear $\mathscr{E}_n$
```

6.1. *Marking faces to determine required layers*

The process described in Algorithm 12 is performed in three stages:

Creating lists and maps. From lines 1 to 3, the procedure creates an empty list of faces $\mathscr{L}$ and two empty maps $\mathscr{E}_c$ and $\mathscr{E}_n$. The list $\mathscr{L}$ is used during the execution to mark the faces as layers. Both structures, $\mathscr{E}_c$ and $\mathscr{E}_n$, are used to map an indexed edge to a list of adjacent faces. Specifically, $\mathscr{E}_c$ and $\mathscr{E}_n$ store the current and the new fronts, respectively.

Creating initial fronts. From lines 4 to 6, the algorithm indexes the boundary edges that are on a corner or touch a vertex with valence greater than three. For each indexed edge e its adjacent boundary faces are stored in list $\mathscr{E}_c[e]$. Thus, the initial front is determined by indexed boundary edges and their adjacent boundary faces stored in $\mathscr{E}_c$.

Advancing fronts. From lines 9 to 21, new faces are marked as layers in an advancing front manner. The fronts are advanced using the inner faces adjacent to the indexed edges in $\mathscr{E}_c$. Those inner faces that have similar normal directions to the faces stored in indexed lists of $\mathscr{E}_c$ are marked as layers. That is, for each front face we select the adjacent inner face that minimizes the cosine of the inner angle between the normal vectors of both faces. These new layer faces and its inner edges not in $\mathscr{E}_c$ determine a new front stored in $\mathscr{E}_n$. The fronts are advanced whereas new faces are marked as layers. That is, the procedure stops when all fronts again reach the boundaries.

To illustrate this advancing front procedure, in Figure 19 we present the three first steps around a corner curve. In Figure 19(a), each one of the initial front edges, dashed lines, has a list of two

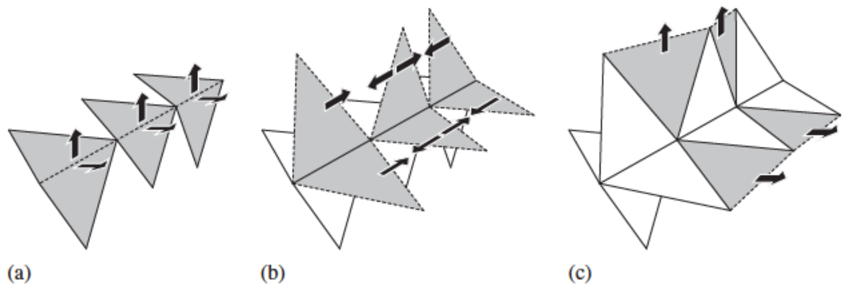

Figure 19. Advancing fronts of edges and faces from a corner curve: (a) initial front; (b) new front at first iteration; and (c) new front at second iteration.

adjacent boundary faces, gray triangles. The arrows indicate the preferred advancing directions. Figure 19(b) shows the optimal faces, gray triangles, adjacent to initial front edges and the new front edges, dashed lines. These new faces and edges compose the new front. The next iteration of the advancing front loop is presented in Figure 19(c). A new front is determined by new optimal inner faces, gray triangles, and inner edges not in the previous front, dashed lines.

6.2. Dealing with 4-valent vertices

It is well known that discretizing with hexahedra the region around a feature vertex with valence greater than three is a difficult task. However, it is straightforward to automatically discretize these regions with tetrahedra. This is one of the advantages of using a tetrahedral mesh as a reference mesh for adding local dual contributions. Note that all nodes in a tetrahedron and in a hexahedron are 3-valent. Therefore, for each feature vertex with valence greater than three the reference mesh is naturally decomposed in 3-valent vertices. Algorithm 12 takes into account this reasoning. In particular, according to line 5 of Algorithm 12 we propose to add to the initial front those edges that touch a vertex with valence greater than three. Then, the front advances through the reference mesh and the obtained layer faces naturally divide the feature vertex into several 3-valent vertices.

7. EXAMPLES

In this section we present several examples to illustrate the capabilities of the proposed approach. The local dual contributions allow capturing the dual of the desired block mesh for a wide range of convex and non-convex domains. The examples present geometric features including planar surfaces, curved surfaces, vertices with valence greater than three, thin configurations, and holes.

In all examples we have used Tetgen package [19] to generate the reference mesh. However, additional vertices are added on several curves of the geometry to guide the generation of the reference mesh. Figures 21(a) and 22(a) depict the location of these additional vertices (black dots) for the corresponding geometries.

We use our implementation to decompose these geometries in coarse blocks. Notice that the proposed algorithm focuses on the generation of valid topological decomposition in blocks. Then, the final node location can be improved using a relaxation procedure. In addition, a finer mesh can be obtained by meshing each block separately while keeping the compatibility of the mesh between them.

For the first three examples, we present four figures showing the geometry to decompose, the representation of the dual surfaces with local dual contributions, the dual regions delimited by the dual surfaces and the obtained block mesh. For the last four examples, which contain a large number of local dual contributions, we only include the dual regions and the associated block mesh. In all the examples, the local dual contributions are colored according to their type (colors are available in the online version): hard face local dual contributions (triangles) are colored in red; edge candidates (long quadrilaterals) are colored in orange and cyan to indicate hard and soft local dual contributions, respectively; and vertex candidates (small quadrilaterals) are colored in yellow and blue to indicate hard and soft local dual contributions, respectively. The dual regions are also depicted in different shades: each shaded region is associated with a different primal node of the block mesh. Furthermore, for each example we include a table with the number of mesh nodes and elements that compose the reference mesh, the dual surfaces represented with local dual contributions, the dual regions, and the final block mesh. Note that the number of polygons that define the dual surfaces is at most 28 times the number of reference elements. On the contrary, the number of polyhedra that determine the dual regions is always 15 times the number of reference elements.

Brick-shaped domain. This geometry is a convex domain that is tapered in two orthogonal directions, see Figure 20(a). All the dual surfaces follow the boundary of the domain and the possible gaps between local dual contributions are filled with soft local dual contributions, see Figure 20(b). Hard local dual contributions from edges and vertices determine the desired mesh close to feature curves and vertices. The resulting local dual contributions are a discrete representation of the dual surfaces and divide the domain into several dual regions represented in different shades in Figure 20(c). Each one of the dual regions corresponds to a primal node of the final block mesh, see Figure 20(d). In this example, all the vertices are 3-valent. Hence, it can be meshed with submapping or midpoint subdivision. Specifically, the obtained decomposition is equivalent to a coarse mesh obtained with these methods. Table II shows the number of nodes and elements that compose the meshes used to obtain the final block mesh.

L-shaped domain. This geometry is non-convex and presents a curved surface determined by a rounding, see Figure 21(a). Our meshing tool automatically adds local dual contributions that follow the surface boundaries and that cut the domain along the corner curve, see Figure 21(b). These local dual contributions divide the domain into several dual regions represented in different shades in Figure 21(c). Each one of the dual regions corresponds to a primal node of the final block mesh, see Figure 21(d). Note that to generate a hexahedral mesh for this domain we can use sweeping but not submapping. Table III shows the number of nodes and elements that compose the meshes used to obtain the final block mesh.

Domain with non-sweepable protrusions. This non-convex domain presents two transversal and non-sweepable protrusions. The block meshing tool can decompose the domain with the desired unstructured blocks, see Figure 22(a). Note that neither submapping nor sweeping techniques can generate this decomposition. The mesher adds local dual contributions to the reference mesh, see Figure 22(b), to determine a valid arrangement of dual surfaces. This arrangement delimits several dual regions, see Figure 22(c), which determine the final block mesh, see Figure 22(d). Table IV shows the number of entities used in the meshing process.

Half of a gear. This non-convex domain presents a large number of corner curves and protrusions around the revolution axis. The shape and the dual regions are depicted in Figure 23(a).

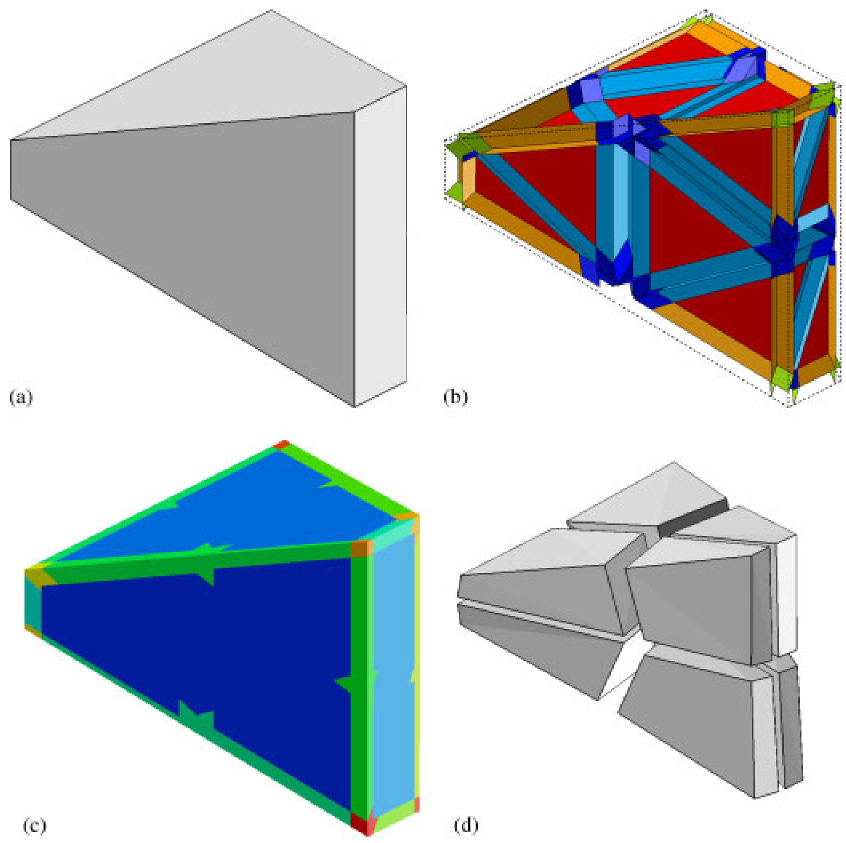

Figure 20. Block-meshing process for the brick-shaped domain: (a) domain; (b) local dual contributions; (c) dual regions; and (d) unstructured block mesh.

Table II. Mesh entities used for the brick-shaped domain.

	Nodes	Elements	Type
Reference mesh	16	21	Tetrahedra
Dual surfaces	336	406	Polygons
Dual regions	368	315	Polyhedra
Block mesh	27	8	Blocks

The different dual regions split the domain around non-convex curves. The resulting block mesh, see Figure 23(b), is similar to a coarse mesh obtained with the sweeping and submapping techniques. Owing to the number of corner curves we obtain a great amount of local dual contributions, see Table V.

Thin domain with holes. This non-convex domain is curved and has two square-shaped holes. The dual regions, see Figure 24(a), determine two stretched layers of hexahedra through its thickness,

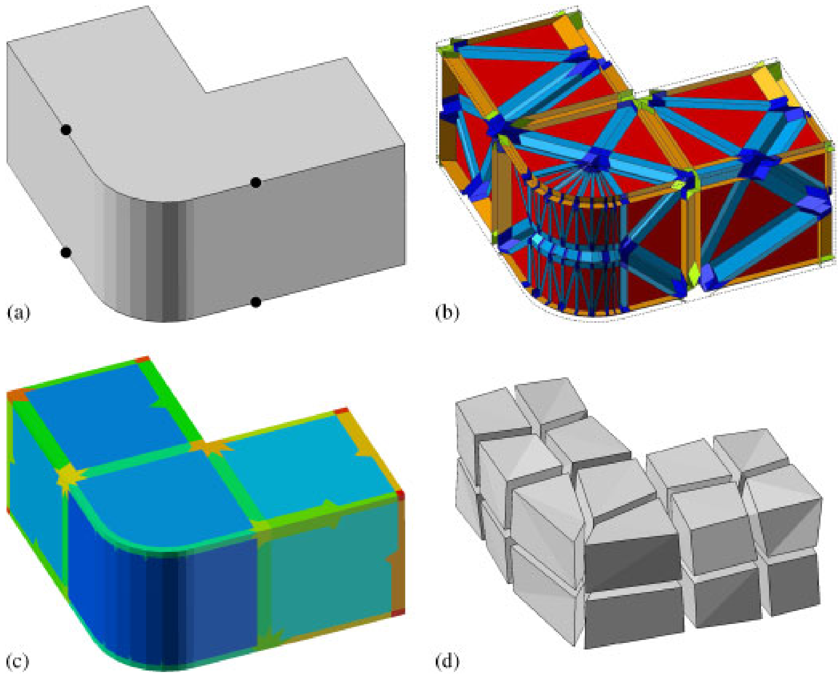

Figure 21. Block-meshing for the L-shaped domain: (a) domain; (b) local dual contributions; (c) dual regions; and (d) unstructured block mesh.

Table III. Mesh entities used for the L-shaped domain.

	Nodes	Elements	Type
Reference mesh	52	88	Tetrahedra
Dual surfaces	1198	1374	Polygons
Dual regions	1460	1320	Polyhedra
Block mesh	50	22	Blocks

see Figure 24(b). As it is a curved domain a large number of boundary faces are needed to represent the shape. Hence, a large number of local dual contributions are needed, see Table VI.

Closed loop. This example presents a non-convex domain that can be meshed with neither submapping nor sweeping. Moreover, this example requires a fully unstructured hexahedral generator. Figure 25(a) shows the obtained dual regions that lead to the final unstructured block mesh, see Figure 25(b). The number of mesh entities used by the algorithm is presented in Table VII.

4-valent protrusion domain. A parallelepiped base with a pyramidal protrusion composes the last example. Herein the difficulty to mesh with hexahedral elements is that the top vertex is 4-valent, see Figure 26(a). Our block mesher splits the space around this feature vertex by following the reference mesh in an advancing front manner, see Figure 26(b). The resulting division allows inserting the 3-valent hexahedral elements. Note that the cuts are propagated along the whole of the domain. The number of entities used along the meshing process is summarized in Table VIII.

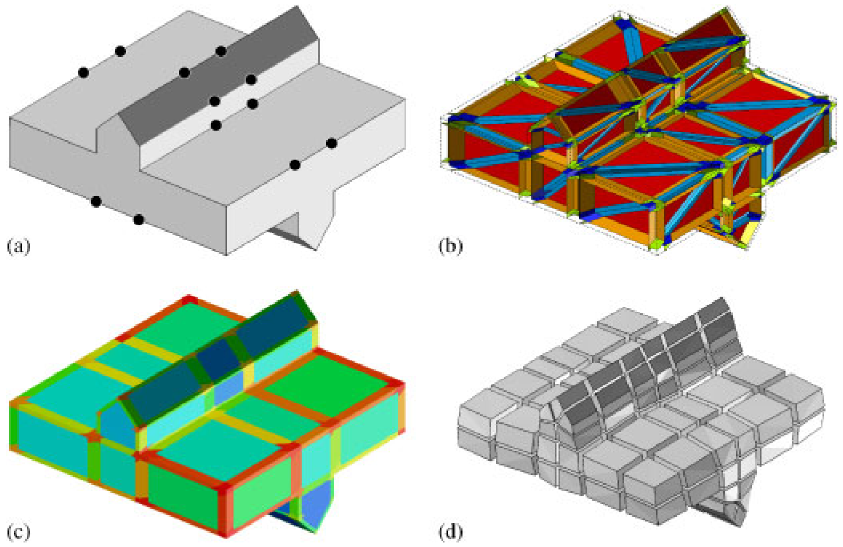

Figure 22. Block-meshing for the domain with non-sweepable protrusions: (a) domain; (b) local dual contributions; (c) dual regions; and (d) unstructured block mesh.

Table IV. Mesh entities used for the domain with non-sweepable protrusions.

	Nodes	Elements	Type
Reference mesh	57	114	Tetrahedra
Dual surfaces	1750	1374	Polygons
Dual regions	1807	1710	Polyhedra
Block mesh	260	132	Blocks

8. CONCLUDING REMARKS AND FUTURE WORK

In this work we have proposed and detailed a new tool, the local dual contributions, for directly representing the topology and the geometry of a block mesh dual. Specifically, the main contribution of this work is to detail how to add local dual contributions on a reference mesh to represent the desired dual configurations. To this end, we present a hierarchical scheme and a set of matching rules to ensure that we obtain a valid dual configuration. That is, the local dual contributions define a dual of the block mesh with intersections of the proper multiplicity, without gaps and which reproduce the boundary features of the domain.

Practical results suggest that the proposed types of local dual contributions, hard and soft, can capture the dual of the desired block mesh for a wide range of geometries. In particular, soft local dual contributions allow obtaining the same local dual configuration for different reference meshes of the domain.

The computational cost of adding local dual contributions is the time of creating a tetrahedral reference mesh plus the time of checking all possible candidates to contribute to the dual. Note that tetrahedral mesh generators are efficient and the number of candidates is 28 times the number

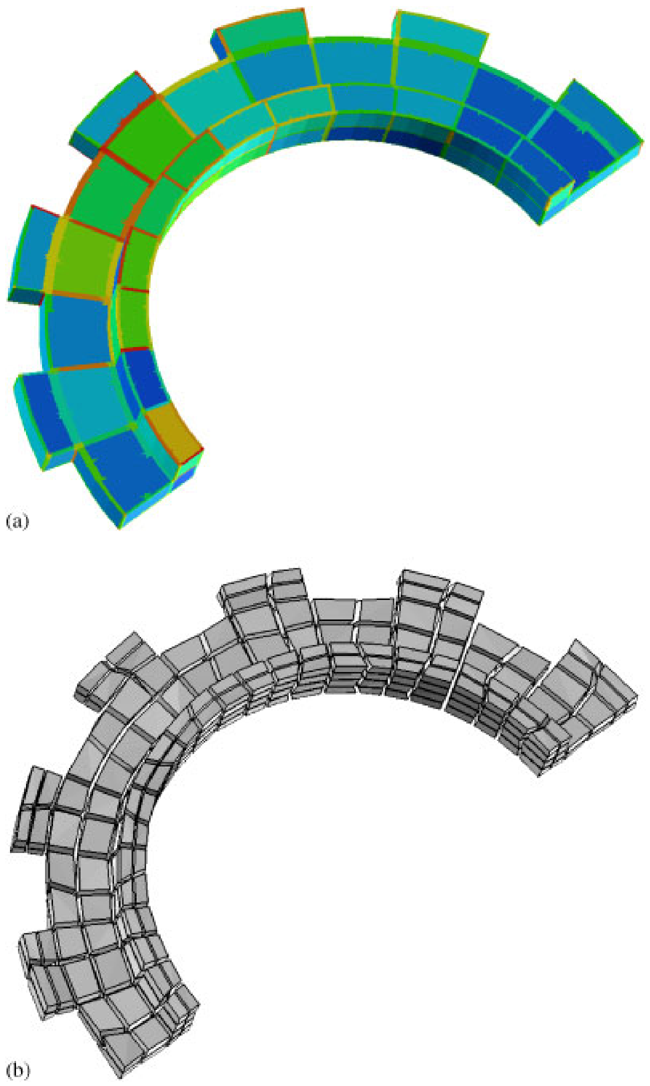

Figure 23. Block-meshing for the half of a gear domain: (a) dual regions and (b) unstructured block mesh.

of reference elements. Hence, the local dual contributions tool has an asymptotic behavior equivalent to the used tetrahedral mesh generator. Therefore, no effort has been made to optimize the performance of our implementation. Nevertheless, the performance of the current implementation can be improved through code optimization.

This tool allows us to state a novel approach to generate block decompositions of a given domain when there is no prescribed boundary mesh. The proposed algorithm is developed in three steps: (i) the generation of a coarse reference mesh composed by tetrahedral elements; (ii) the insertion of dual surfaces by addition of local dual contributions; and (iii) the dual regions are

Table V. Mesh entities used for the half of a gear example.

	Nodes	Elements	Type
Reference mesh	513	1229	Tetrahedra
Dual surfaces	16697	20966	Polygons
Dual regions	18840	18435	Polyhedra
Block mesh	633	336	Blocks

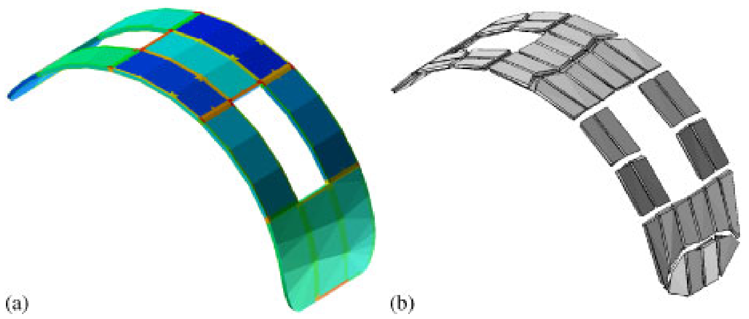

Figure 24. Block-meshing for the thin domain with holes: (a) dual regions and (b) unstructured block mesh.

Table VI. Mesh entities used for the thin domain with holes.

	Nodes	Elements	Type
Reference mesh	391	1052	Tetrahedra
Dual surfaces	13445	15657	Polygons
Dual regions	15764	15780	Polyhedra
Block mesh	201	96	Blocks

dualized to obtained the final block mesh. We have implemented a block mesh generator to show a first application of this scheme. To this end, we have presented an automatic procedure to mark inner reference faces as layers. Several examples show that the current implementation generates the expected decomposition for several convex and non-convex blocky geometries. Moreover, we obtain the desired unstructured decomposition in blocks even in those examples where submapping and sweeping techniques fail. These examples present different geometrical characteristics, such as planar surfaces, curved surfaces, thin configurations, holes, and vertices with valence greater than three. Specifically, the tool deals with 4-valent vertices due to the use of a reference mesh composed by tetrahedra. The tetrahedral elements provide a natural splitting of the high-valence vertices in 3-valent nodes.

Obtaining a general-purpose block mesh generator is our long-term goal. To this end, we have proposed the new concept of local dual contributions. In this sense, this work presents several issues that should be investigated and solved in the near future. First, the quality of the final block decomposition depends on the reference mesh. To mitigate this dependence, we claim that the reference mesh has to be modified while the fronts of marked faces advance through the geometry.

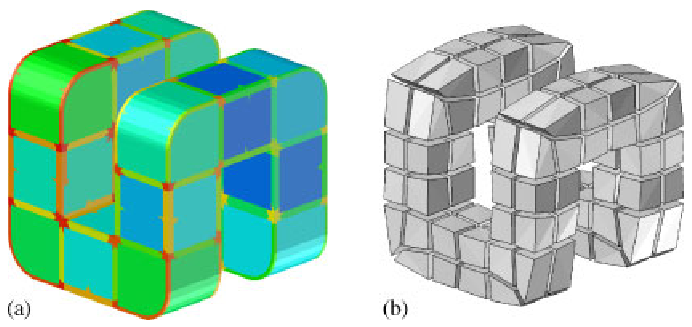

Figure 25. A rounded and closed loop geometry: (a) dual regions and (b) unstructured block mesh.

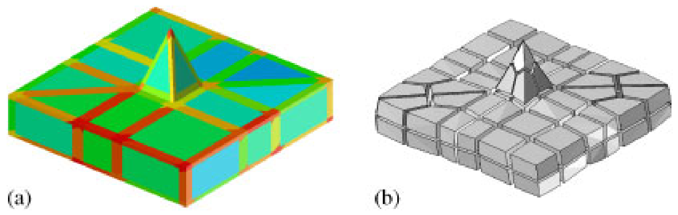

Figure 26. Geometry with a 4-valent protrusion (pyramid): (a) dual regions and (b) unstructured block mesh.

Table VII. Mesh entities used for the rounded loop example.

	Nodes	Elements	Type
Reference mesh	303	579	Tetrahedra
Dual surfaces	7527	8626	Polygons
Dual regions	9372	8685	Polyhedra
Block mesh	240	112	Blocks

Table VIII. Mesh entities used for the 4-valent protrusion example.

	Nodes	Elements	Type
Reference mesh	33	56	Tetrahedra
Dual surfaces	891	1312	Polygons
Dual regions	924	840	Polyhedra
Block mesh	177	92	Blocks

Second, we have to reduce the propagation through the mesh of the cuts that start at 4-valent vertices. Third, we have to extend the proposed method to deal with high-valence vertices (5 or 6-valent vertices). Fourth, additional logic might be incorporated to apply the proposed approach to more complicated geometries such as assembly models (contiguous domains with shared curves and surfaces) and non-blocky geometries (smooth domains without sharp features, such as spheres or cylinders). Fifth, we have to analyze if a fine reference mesh, adapted to a prescribed element size, can be used to generate hexahedral meshes by means of adding local dual contributions. Finally, we realize that the current approach adds local dual contributions on the potential set of all the polygon candidates of an initial dual arrangement. However, we are analyzing the alternative approach of first adding all the polygon candidates as hard. This is equivalent to obtaining a tet-to-hex mesh. Then, this initial dual can be modified iteratively to obtain a better dual configuration according to a heuristic that controls the quality of the dual. To this end, several local dual contributions can be removed or converted to soft local dual contributions at each iteration.

APPENDIX A

A.1. Intersection points for planar dual surfaces

Here we list the barycentric coordinates of the intersection points for the planar dual surfaces in a reference element. Let μ be a fixed number such that $0<\mu<\frac{1}{4}$. This number determines the position of the planar dual surfaces inside a reference element. The 12 intersection points at the reference edges are:

$$\begin{aligned}
&p_{12}=(\lambda_e,\mu,0,0), && p_{21}=(\mu,\lambda_e,0,0), && p_{13}=(\lambda_e,0,\mu,0), && p_{31}=(\mu,0,\lambda_e,0),\\
&p_{14}=(\lambda_e,0,0,\mu), && p_{41}=(\mu,0,0,\lambda_e), && p_{23}=(0,\lambda_e,\mu,0), && p_{32}=(0,\mu,\lambda_e,0),\\
&p_{24}=(0,\lambda_e,0,\mu), && p_{42}=(0,\mu,0,\lambda_e), && p_{34}=(0,0,\lambda_e,\mu), && p_{43}=(0,0,\mu,\lambda_e),
\end{aligned}$$

where $\lambda_e:=1-\mu$. Note that the first index i of an intersection point p_{ij} determines the position of λ_e in the barycentric coordinates. The twelve intersection points at the reference faces are:

$$\begin{aligned}
&p_{234}=(0,\lambda_f,\mu,\mu), && p_{324}=(0,\mu,\lambda_f,\mu), && p_{423}=(0,\mu,\mu,\lambda_f),\\
&p_{413}=(\mu,0,\mu,\lambda_f), && p_{314}=(\mu,0,\lambda_f,\mu), && p_{134}=(\lambda_f,0,\mu,\mu),\\
&p_{142}=(\lambda_f,\mu,0,\mu), && p_{214}=(\mu,\lambda_f,0,\mu), && p_{412}=(\mu,\mu,0,\lambda_f),\\
&p_{312}=(\mu,\mu,\lambda_f,0), && p_{213}=(\mu,\lambda_f,\mu,0), && p_{123}=(\lambda_f,\mu,\mu,0),
\end{aligned}$$

where $\lambda_f:=1-2\mu$. Note that the first index i of an intersection point p_{ijk} determines the position λ_f in the barycentric coordinates. The four inner intersection points are:

$$p_{1234}=(\lambda_t,\mu,\mu,\mu),\quad p_{2134}=(\mu,\lambda_t,\mu,\mu),\quad p_{3124}=(\mu,\mu,\lambda_t,\mu),\quad p_{4123}=(\mu,\mu,\mu,\lambda_t),$$

where $\lambda_t:=1-3\mu$. Note that the first index i of an intersection point p_{ijkl} determines the position λ_t in the barycentric coordinates.

A.2. All candidate polygons

In this section we list the 28 possible candidates inside a reference element the: face candidates, edge candidates, and vertex candidates. All the candidates are expressed according to the intersection points of the planar dual surfaces, see Section A.1. The four possible face candidates are expressed in terms of the inner intersection points:

$$\mathbf{c}(f_{234};t)=[p_{2134},p_{3124},p_{4123}],\quad \mathbf{c}(f_{134};t)=[p_{4123},p_{3124},p_{1234}],$$
$$\mathbf{c}(f_{124};t)=[p_{1234},p_{2134},p_{4123}],\quad \mathbf{c}(f_{123};t)=[p_{3124},p_{2134},p_{1234}].$$

The 12 edge candidates are expressed in terms of inner and face intersection points:

$$\mathbf{c}(e_{12};f_{123};t)=[p_{213},p_{123},p_{1234},p_{2134}],\quad \mathbf{c}(e_{12};f_{124};t)=[p_{124},p_{214},p_{2134},p_{1234}],$$
$$\mathbf{c}(e_{13};f_{134};t)=[p_{314},p_{134},p_{1234},p_{3124}],\quad \mathbf{c}(e_{13};f_{123};t)=[p_{123},p_{312},p_{3124},p_{1234}],$$
$$\mathbf{c}(e_{14};f_{134};t)=[p_{134},p_{413},p_{4123},p_{1234}],\quad \mathbf{c}(e_{14};f_{124};t)=[p_{412},p_{124},p_{1234},p_{4123}],$$
$$\mathbf{c}(e_{23};f_{234};t)=[p_{234},p_{324},p_{3124},p_{2134}],\quad \mathbf{c}(e_{23};f_{123};t)=[p_{312},p_{213},p_{2134},p_{3124}],$$
$$\mathbf{c}(e_{24};f_{234};t)=[p_{423},p_{234},p_{2134},p_{4123}],\quad \mathbf{c}(e_{24};f_{124};t)=[p_{214},p_{412},p_{4123},p_{2134}],$$
$$\mathbf{c}(e_{34};f_{234};t)=[p_{324},p_{423},p_{4123},p_{3124}],\quad \mathbf{c}(e_{34};f_{134};t)=[p_{413},p_{314},p_{3124},p_{4123}].$$

The 12 vertex candidates are expressed in terms of inner, face, and edge intersection points:

$$\mathbf{c}(v_1;e_{12};t)=[p_{12},p_{123},p_{1234},p_{124}],\quad \mathbf{c}(v_1;e_{13};t)=[p_{13},p_{123},p_{1234},p_{134}],$$
$$\mathbf{c}(v_1;e_{14};t)=[p_{14},p_{124},p_{1234},p_{134}],\quad \mathbf{c}(v_2;e_{12};t)=[p_{21},p_{213},p_{2134},p_{214}],$$
$$\mathbf{c}(v_2;e_{23};t)=[p_{23},p_{213},p_{2134},p_{234}],\quad \mathbf{c}(v_2;e_{24};t)=[p_{24},p_{214},p_{2134},p_{234}],$$
$$\mathbf{c}(v_3;e_{13};t)=[p_{31},p_{312},p_{3124},p_{314}],\quad \mathbf{c}(v_3;e_{23};t)=[p_{32},p_{312},p_{3124},p_{324}],$$
$$\mathbf{c}(v_3;e_{34};t)=[p_{34},p_{314},p_{3124},p_{324}],\quad \mathbf{c}(v_4;e_{14};t)=[p_{41},p_{412},p_{4123},p_{413}],$$
$$\mathbf{c}(v_4;e_{24};t)=[p_{42},p_{412},p_{4123},p_{423}],\quad \mathbf{c}(v_4;e_{34};t)=[p_{43},p_{413},p_{4123},p_{423}].$$

REFERENCES

1. Blacker TD. Automated conformal hexahedral meshing constraints, challenges and opportunities. *Engineering with Computers* 2001; **17**(3):201–210.
2. Tautges TJ. The generation of hexahedral meshes for assembly geometry: survey and progress. *International Journal for Numerical Methods in Engineering* 2001; **50**(12):2617–2642.
3. Carey GF. Hexing the tet. *Communications in Numerical Methods in Engineering* 2002; **18**(3):223–227.
4. Tautges TJ, Blacker TD, Mitchell SA. The whisker weaving algorithm: a connectivity-based method for constructing all-hexahedral finite element meshes. *International Journal for Numerical Methods in Engineering* 1996; **39**(19):3327–3350.
5. Mueller-Hannemann M. Hexahedral mesh generation by successive dual cycle elimination. *Engineering with Computers* 1999; **15**(3):269–279.
6. Calvo NA, Idelsohn SR. All-hexahedral element meshing: generation of the dual mesh by recurrent subdivision. *Computer Methods in Applied Mechanics and Engineering* 2000; **182**(3–4):371–378.
7. Staten ML, Owen SJ, Blacker TD. Unconstrained paving and plastering: a new idea for all hexahedral mesh generation. *14th International Meshing Roundtable*, San Diego, 2005; 399–416.
8. Staten ML, Kerr RA, Owen SJ, Blacker TD, Stupazzini M, Shimada K. Unconstrained plastering—hexahedral mesh generation via advancing-front geometry decomposition. *International Journal of Numerical Methods in Engineering*, 2009; **81**(2):135–171.

9. Price M, Armstrong C, Sabin M. Hexahedral mesh generation by medial surface subdivision: part I. Solids with convex edges. *International Journal for Numerical Methods in Engineering* 1995; **38**(19):3335–3359.
10. Price MA, Armstrong CG. Hexahedral mesh generation by medial surface subdivision: part II. Solids with flat and concave edges. *International Journal for Numerical Methods in Engineering* 1997; **40**(1):111–136.
11. Sheffer A, Etzion M, Rappoport A, Bercovier M. Hexahedral mesh generation using the embedded Voronoi graph. *Engineering with Computers* 1999; **15**(3):248–262.
12. Sheffer A, Bercovier M. Hexahedral meshing of non-linear volumes using Voronoi faces and edges. *International Journal for Numerical Methods in Engineering* 2000; **49**:329–351.
13. Murdoch P, Benzley S, Blacker TD, Mitchell SA. The spatial twist continuum: a connectivity based method for representing all-hexahedral finite element meshes. *Finite Elements in Analysis and Design* 1997; **28**(2):137–149.
14. Calvo NA. Generación de mallas tridimensionales por métodos duales. *Ph.D. Thesis*, Universidad Nacional del Litoral, 2005.
15. Mitchell SA. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of enclosed volume. *Lecture Notes in Computer Science* 1996; **1046**:465–478.
16. Blacker TD, Stephenson MB. Paving: a new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering* 1991; **32**(4):811–847.
17. Whiteley M, White DR, Benzley S, Blacker TD. Two and three-quarter dimensional meshing facilitators. *Engineering with Computers* 1996; **12**(3–4):144–154.
18. Shepherd JF, Johnson CR. Hexahedral mesh generation constraints. *Engineering with Computers* 2008; **24**(3): 195–213.
19. Si H. TetGen. *A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator*: *Users's Manual for Version*, vol. 1, 2006. Available from: http://tetgen.berlios.de.