Hexahedral-Dominant Meshing

Dmitry Sokolov, Nicolas Ray, Lionel Untereiner, Bruno Lévy

Hexahedral-dominant meshing

Dmitry Sokolov
Université de Lorraine
and
Nicolas Ray
INRIA Nancy Grand-Est
and
Lionel Untereiner
INRIA Nancy Grand-Est
and
Bruno Lévy
INRIA Nancy Grand-Est

This article introduces a method that generates a hexahedral-dominant mesh from an input tetrahedral mesh. It follows a three-steps pipeline similar to the one proposed by Carrier Baudoin *et al.*: (1) generate a frame field; (2) generate a pointset P that is mostly organized on a regular grid locally aligned with the frame field; and (3) generate the hexahedral-dominant mesh by recombining the tetrahedra obtained from the constrained Delaunay triangulation of P.

For step (1), we use a state of the art algorithm to generate a smooth frame field. For step (2), we introduce an extension of Periodic Global Parameterization to the volumetric case. As compared with other global parameterization methods (such as CubeCover), our method relaxes some global constraints to avoid creating degenerate elements, at the expense of introducing some singularities that are meshed using non-hexahedral elements. For step (3), we build on the formalism introduced by Meshkat and Talmor, fill-in a gap in their proof and provide a complete enumeration of all the possible recombinations, as well as an algorithm that efficiently detects all the matches in a tetrahedral mesh.

The method is evaluated and compared with the state of the art on a database of examples with various mesh complexities, varying from academic examples to real industrial cases. Compared with the method of Carrier-Baudoin *et al.*, the method results in better scores for classical quality criteria of hexahedral-dominant meshes (hexahedral proportion, scaled Jacobian, etc.). The method also shows better robustness than CubeCover and its derivatives when applied to complicated industrial models.

INRIA Team ALICE provided support.
Authors' addresses: Dmitry.Sokolov@univ-lorraine.fr, Nicolas.Ray@inria.fr, Lionel.Untereiner@inria.fr, Bruno.Levy@inria.fr

DOI 10.1145/XXXXXXX.YYYYYYY
http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically based modeling*

General Terms: Algorithms

Additional Key Words and Phrases: Hexahedral-dominant remeshing, mesh processing, global parameterization

ACM Reference Format:

1. INTRODUCTION

We propose a new hexahedral-dominant meshing algorithm, that is to say an algorithm that creates from an input tetrahedral mesh a new mesh where most cells are hexahedral. We use the three steps pipeline (Fig. 1) introduced in [Carrier-Baudouin et al. 2014] and used in [Botella et al. 2015]: (1) create a frame field to steer the placement and orientation of the cells; (2) generate a pointset P that mostly corresponds to the vertices of a grid aligned with the frame field, (3) merge the tetrahedra in the Delaunay triangulation of P in order to create hexahedra. Our contribution is two-fold: the pointset is generated by global optimization (instead of the front propagation used in [Carrier-Baudouin et al. 2014] and [Botella et al. 2015]), and we complement our initial intuition outlined in [Botella et al. 2015] with a thorough analysis of the recombination problem (merging tetrahedra into hexahedra), leading to a both robust and efficient algorithm.

We compute the input frame fields using [Ray and Sokolov 2015], and define the point set by extracting the intersections of integer-valued iso-surfaces of a global parameterization. A global parameterization is required everywhere in the volume for full hex meshing [Nieser et al. 2011; Li et al. 2012; Jiang et al. 2014], but *hexahedral-dominant* remeshing can deal with incomplete global parameterizations. As a consequence, point sets can be generated without enforcing constraints in the frame field by combinatorial / integer variable optimization (without guarantee) as in [Li et al. 2012; Jiang et al. 2014]. In our case, the global parameterization is computed by a volumetric extension of Periodic Global Param-

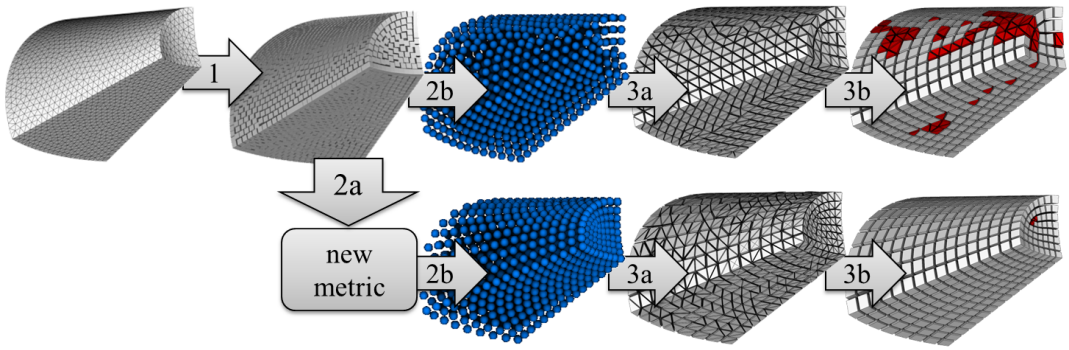


Fig. 1: Starting from a tetrahedral mesh, we compute a frame field (1), optionally optimize the metric to increase the proportion of hexahedra (2a), compute a pointset (2b), produce a new tetrahedral mesh with the pointset as vertices and (3a) generate the hexahedral-dominant mesh by recombining the tetrahedra (3b).

eterization [Ray et al. 2006] that can naturally make singularities emerge.

To extract the hexahedral-dominant mesh from the global parameterization, we first generate a pointset P by mapping the points with integer-valued parameters. The points in P are then interconnected with tetrahedra using the Delaunay triangulation of P constrained to the boundary of the domain. We finally recombine the tetrahedra into hexahedra, with an algorithm that extends the approach of [Meshkat and Talmor 2000]. We fill-in a gap in the original proof, extend the analysis to all the configurations with slivers, and identify some non-trivial forbidden configurations that are ruled-out by our algorithm.

In most cases, our method outperforms the best hexahedral-dominant method [Carrier-Baudouin et al. 2014] both in proportion and quality of hexahedral elements. This is mostly due to the front propagation approach they use, that creates discontinuities close to the medial axis. Compared with full hexahedral remeshing [Li et al. 2012], the main advantage of our method is its ability of handling more industrial-size difficult cases (at the expense of introducing non hexahedral elements). The main drawback of our approach is that it may introduce non-hexahedral elements and/or non-natural branching structures, even in simple cases, where a full hexahedral mesher can avoid them and generate a more structured mesh, with for instance a constant number of layers of hexahedra (Fig. 21).

Pipeline overview

Our method is summarized in Fig. 1. Starting from an input tetrahedral mesh, it consists in the following three steps:

(1) **Generate the frame field that specifies the desired size and orientation of the elements** We use the algorithm in [Ray and Sokolov 2015], and set the desired edge size to be equal to the average edge size of the original mesh. It is also possible to explicitly specify the desired scale by a function (§4.2.3). *This step uses state-of-the-art methods.*

(2) **Generate the pointset** P
- —Modify the prescribed size and orientations in order to increase the proportion of hexahedra in the final mesh (§2.4), as shown in Fig. 1-bottom *(this substep is optional).*
- —Generate an atlas of grid-compatible maps (§2.2).
- —Extract the pointset (§2.3).

(3) **Generate the hexahedral dominant mesh**
- —Remesh the border of the domain using the points in P (§3.1);
- —generate a tetrahedral mesh using the Delaunay triangulation of the pointset P constrained by the remeshed border. *This substep uses state-of-the-art methods* [George et al. 1990], [Si 2015].
- —Find all the candidate hexahedra that can be obtained by merging tetrahedra (§3.2.3).
- —Select among them the best mutually compatible set of hexahedra (§3.2.4).

Previous work

The state of the art in tetrahedral meshing has now reached a maturity that makes it reasonably easy to mesh arbitrary shapes using existing software [George et al. 1990; Si 2015; CGAL] . . . For hexahedral meshing, the situation is different, and despite important advances, the state of the art is still far away from a general and robust fully automatic solution. The number of failure cases remains important, even for simple objects that can exhibit some difficult combinatorial aspects of the problem. Despite an important amount of research effort to solve these issues, designing a complete hexahedral remeshing algorithm requires to solve many open problems [Shepherd and Johnson 2008]. For this reason, hexahedral-dominant meshing may be an option worth investigating: by relaxing the problem, it still generates a valid result in complicated cases where full-hexahedral methods generally fail, at the expense of introducing non-hexahedral elements. Depending on the targeted application (e.g., Finite Element simulation), some element types will be prefered (only hexahedra and tetrahedra, or also pyramid and prisms). Besides multiple element types, the used Finite Element software needs to be able to handle non-conformal junctions (i.e. between a hexahedron and two tetrahedra). In a certain sense, hexahedral dominant meshing "pushes" some of the difficulties towards the Finite Element formulation, which needs defining a well-adapted function space [Bergot et al. 2009]. The results reported in the latter article show the efficiency of hybrid meshes in several different contexts.

Hexahedral-dominant meshing is often treated by adapting a hexahedral-only meshing method and completing the result with tetrahedra whenever the method gets stuck in configurations that cannot be meshed with hexahedra only. It is also recommended not to use it as a full automatic process [Meyers et al. 1998]. Our strategy to tackle the problem is different: we first distribute points in-

side the volume to be meshed, then generate a tetrahedral mesh having these points as vertices, and finally form hexahedra by recombining tetrahedra from this mesh. This strategy decomposes the problem into two subproblems: **(1) Generating the points:** generate points that are likely to be the vertices of a nice hexahedral-dominant mesh and **(2) Recombining the tetrahedra into hexahedra:** finding the best hexahedra that can be recombined within a tetrahedral mesh.

(1) **Generating the points:** In previous works, points were distributed by a centroidal Voronoi with a norm L_p in [Lévy and Liu 2010], or by a front propagation approach as used in [Carrier-Baudouin et al. 2014]. We introduce another strategy inspired by remeshing using a global parameterization.
In $2D$, for quadrilateral remeshing of a triangulated surface, it is possible to define a cross field over the surface (4 orthogonal unit vectors of the tangent plane), then define a special atlas of the surface such that the pre-image of a unit grid defined in the maps gives a quadrilateral mesh of the surface. The first $2D$ method was introduced in [Ray et al. 2006], it produces quadrilateral meshes with very regular size, but not everywhere on the surface. These areas where the algorithm fails to produce quadrilaterals allow to introduce irregularities in the quad mesh. It corresponds either to singularities of the cross field, or to T-vertices that are required to respect the desired scale of the quads. The following approaches [Kälberer et al. 2007; Bommes et al. 2009] are able to generate a quadrilateral mesh on 100% of the surface, at the expense of creating more distorted quadrilaterals (they do not balance the field's curl by introducing T-vertices).

In $3D$, the algorithm in [Nieser et al. 2011] is a direct extension of the $2D$ [Kälberer et al. 2007]. It can produce very nice results. However, its main restriction is that it assumes to have as input a valid $3D$ frame field that corresponds to a hexahedral mesh, and it may significantly stretch the hexahedra in order to balance the frame field's curl. In many (simple) cases, a frame field can be initialized by [Huang et al. 2011], and locally optimized [Li et al. 2012; Jiang et al. 2014] to produce a field topology that corresponds to a hexahedral mesh. This solution is fast and provides excellent results provided that local modifications of the frame field are sufficient to make it compatible with hexahedral remeshing, and that the stretch of hexahedra induced by the frame field's curl corresponds to the user-desired stretch. Another approach to ensure that the frame field's topology is compatible with remeshing is to forbid having any singularity. This approach yields algorithms based on the generation of polycubes [Gregson et al. 2011; Livesu et al. 2013; Huang et al. 2014].

We instead choose to extend [Ray et al. 2006] to the $3D$ case. As in the $2D$ case, it will not be able to produce hexahedra everywhere inside the volume, but it will produce hexahedra of desired stretch and orientation without any constraint on the frame field. The resulting mesh is hexahedral-dominant (rather than pure hexahedral), it has non-hexahedral elements where the Periodic Global Parameterization has singularities.

(2) **Merging tetrahedra into hexahedra** Several algorithms were proposed to merge tetrahedra into hexahedra within a tetrahedral mesh [Yamakawa and Shimada 2003], [Meshkat and Talmor 2000]. The latter reference provides an abstract formalization that can be used to systematically enumerate all the possible configurations. We give a complete explanation of this formalism, fill-in a gap in the original proof and then elaborate on it to obtain an exhaustive enumeration of all the decompositions of a hexahedron into tetrahedra, together with a complete understanding of the structure behind the set of all possible decompositions: any decomposition of the cube can be deduced by simple operations from six "atomic" configurations. This specific understanding of the problem leads to an algorithm that is both short, efficient and easy to implement.

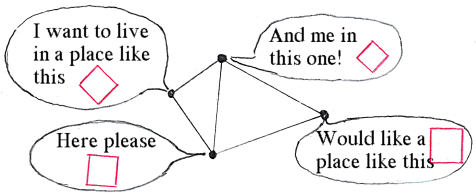


Fig. 2: Each vertex of the input mesh knows the size and the orientation of the hexahedron to be created in his vicinity. We store this information as an orthogonal basis matrix B_i per vertex i.

2. GENERATING THE POINTS

Notations: throughout this section we use bold font to denote vectors. Subscripts are used to index variables. For example, the vertex number i of a tetrahedral mesh will be denoted by $\mathbf{x_i}$. Square brackets are used to access to individual components of vectors or matrices. For instance, $\mathbf{x_i}[0]$ denotes the first component of a three-dimensional vector $\mathbf{x_i}$. Given a matrix R, its first row is denoted by $R[0,\cdot]$ and its first column is denoted by $R[\cdot,0]$.

—**The input of our algorithm** is a tetrahedral mesh T representing an object to be remeshed, as well as the desired sizes and orientations of the hexahedral elements. The desired size and orientation at each vertex i is represented by a matrix B_i such that its columns $B_i[\cdot,k]$, $k=0,1,2$ form an orthogonal basis that corresponds to the edges of the typical hexahedral elements to be created in the vicinity of vertex i (Fig. 2). In our examples, matrices B_is are derived from a smooth $3D$ frame field ;

—**the output of our algorithm** is a set of points P meant to be the vertices of a hexahedral-dominant mesh where the orientation and size of hexahedra is as close as possible to the one defined by the matrices B_i.

Our algorithm consists in the following three steps (an optional step and two mandatory ones):

(1) *Optional step:* optimize the size of the hexahedra to reduce the number of singularities (curl-correction). The curl-corrected frame field is obtained as the solution of a sparse linear system (§2.4);

(2) **Parameterize the object** T by solving a sparse linear system (§2.2).

(3) **Back-project points with integer coordinates** from the parameter space onto the object T (§2.3)

The following subsection details the method that generates the volumetric parameterization. For the sake of clarity, the accompanying illustrations are in $2D$.

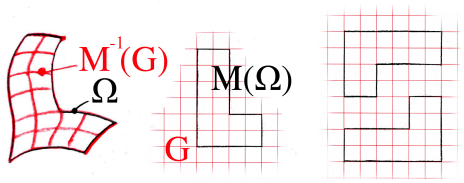


Fig. 3: Considering a bijective continuous map M from an object Ω (left) to $\mathbb{R}^3$ (center), it is possible to generate a hexahedral mesh of Ω by tracing the preimage of the regular grid from $\mathbb{R}^3$ (center) to the object. Note that a whole family of maps – that we call "grid-equivalent" – produces exactly the same hexahedral mesh (right).

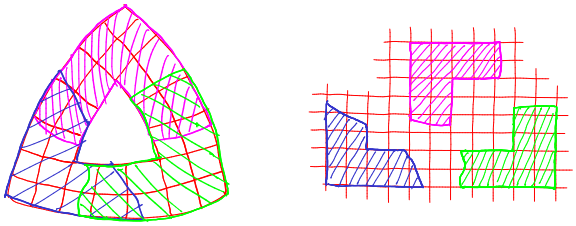


Fig. 4: The object shown on the left can be remeshed with the atlas of grid-equivalent maps shown on the right. The preimage of G through the three maps is unique and defines a hexahedral mesh.

2.1 Problem statement

We suppose that we have a 3D domain Ω and a (global) bijective continuous map $M : \Omega \to \mathbb{R}^3$. Let G denote a regular 3D grid: $G = \{(x, y, z) \in \mathbb{R}^3 : (x \mod 1 = 0)$ or $(y \mod 1 = 0)$ or $(z \mod 1 = 0)\}$. The pre-image $M^{-1}(G)$ of G defines the facets of a hexahedral mesh for Ω (see Fig. 3). In this configuration, $\mathbb{R}^3$ may be thought of as a "texture space", and then the hexahedral mesh of Ω is obtained by using a rectilinear grid as a "texture".

Note that for a given map M and its induced hexahedral mesh $M^{-1}(G)$, it is possible to find another map M' that generates exactly the same mesh (Fig. 3-right). We now formalize such pair of maps that produce the same grid:

DEFINITION 1. *Two maps M and M' are said to be grid-equivalent if and only if there exists $R \in \mathbf{R}$ and $\mathbf{t} \in \mathbf{T}$ such that $M = RM' + \mathbf{t}$, where:*

—$\mathbf{R}$ *is the set of 24 rotation matrices that permute the normal vectors of the facets of the unit cube* $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$, $(-1, 0, 0)$, $(0, -1, 0)$ *and* $(0, 0, -1)$*;*

—$\mathbf{T}$ *is $\mathbb{Z}^3$, the set of vectors with integer coordinates.*

A regular grid G is invariant under the action of the 24 rotations and integer translations: $R \in \mathbf{R}, \mathbf{t} \in \mathbf{T} \Rightarrow G = RG + \mathbf{t}$. Therefore, the preimages of two grid-equivalent maps M and M' correspond to the same hexahedral mesh $M^{-1}(G) = M'^{-1}(G)$ (Fig. 3-right).

In general, a single global map is not sufficient to create a boundary-aligned hexahedral mesh. For example, it is impossible to remesh in this way the object shown on the left of Fig. 4. Thus we will search instead for an *atlas* of local maps. If any pair of maps within the atlas is grid-equivalent on the intersection of their domain, then the final remesh is still unique (see Fig. 4).

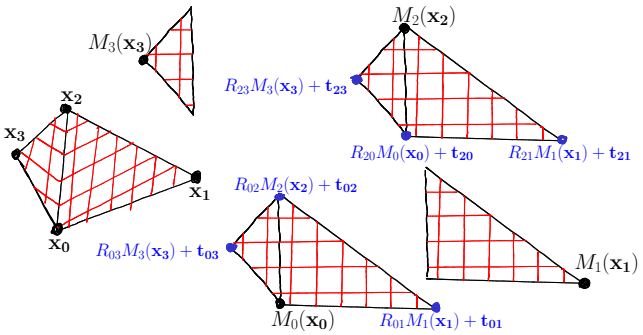


Fig. 5: The maps M_i for the example shown in Fig. 2. Each map M_i is represented by the texture coordinates of the vertices connected to i with an edge. To ensure the grid compatibility of the maps, we constrain the position of the blue vertices: they are obtained from the black ones by an integer translation $\mathbf{t_{ij}}$ (and possibly a rotation).

In our setting, the domain Ω is the tetrahedral mesh T. We associate to each vertex i a local map M_i, defined on the tetrahedra incident to i, and linear in each tetrahedron. We represent each map M_i by texture coordinates of the vertex star of the vertex i.

We formulate our problem as an energy minimization (defined later in this section) under constraints of grid equivalence.

Grid equivalence. Recall that for two adjacent vertices i and j the maps M_i and M_j are grid-equivalent iff there exists $R \in \mathbf{R}$ and $\mathbf{t} \in \mathbf{T}$ such that $M_i = RM_j + \mathbf{t}$. As our maps are linear (per tetrahedron), it is sufficient to enforce this equality at 4 points on each tetrahedron: the maps M_i and M_j are grid-equivalent iff we can find $R_{ij} \in \mathbf{R}$ and $\mathbf{t_{ij}} \in \mathbf{T}$ such that on each tetrahedron (i, j, k, l) we have:

$$\begin{cases} M_i(\mathbf{x_i}) = R_{ij}M_j(\mathbf{x_i}) + \mathbf{t_{ij}} & (1) \\ M_i(\mathbf{x_j}) = R_{ij}M_j(\mathbf{x_j}) + \mathbf{t_{ij}} & (2) \\ M_i(\mathbf{x_k}) = R_{ij}M_j(\mathbf{x_k}) + \mathbf{t_{ij}} & (3) \\ M_i(\mathbf{x_l}) = R_{ij}M_j(\mathbf{x_l}) + \mathbf{t_{ij}} & (4) \end{cases} \quad (1)$$

Thus the problem consists in finding $M_i(\mathbf{x_i})$ and $M_i(\mathbf{x_j})$ for each oriented edge $i < j$, under the constraint of grid-equivalence Eqn (1). It is difficult to find an expression of the grid-equivalence constraint suitable to numerical optimization (i.e. existence of rotations and translations). Therefore, we prefer to introduce the rotations and translations $(R_{ij}, \mathbf{t}_i)$ into the set of variables, together with constraints that connect the texture coordinates on neighboring maps. However, if we directly introduce unknowns R_{ij} and $\mathbf{t_{ij}}$ for each oriented edge $i < j$, it creates much redundancy in the variables. Indeed, from a texture coordinate $M_i(\mathbf{x_i})$, rotation R_{ij} and translation $\mathbf{t_{ij}}$ we can deduce the value of $M_j(\mathbf{x_i})$, because our system must satisfy the grid-equivalence constraint (1). Therefore, we can remove $M_j(\mathbf{x_i})$ (as well as $M_i(\mathbf{x_j})$) from the set of variables.

Let us sum up: each oriented edge $i < j$ involves $M_i(\mathbf{x_i})$, R_{ij} and $\mathbf{t_{ij}}$ as variables (refer to Fig. 5 for an illustration). Note that with this particular choice of variables, lines 1 and 2 of Equation 1 are naturally satisfied, whereas lines 3 and 4 are not. In our optimization process we ignore these constraints. A configuration where they are not satisfied corresponds to a singularity of the frame field or a singularity of the parameterization (more on this in §2.2). This is a deliberate choice: for divergent fields it al-

lows to create good quality hexahedral elements at the expense of introducing Y-junctions in the parameterization (refer to Fig. 6). We can tune the number of singularities by the curl correction step, as described in §2.4.

Energy. We formulate the problem as a least squares problem with a set of equations per oriented edge $i < j$. These equations express the geometric objective, i.e. make the Jacobian matrix $J(M_i)$ as close as possible to ${B_i}^{-1}$. From now on we denote $M_i(\mathbf{x_i})$ as $\mathbf{u_i}$. Each oriented edge $i < j$ introduces two equations in the objective function, thus we have a system of equations to be solved in the least squares sense:

$$\forall i < j, \quad \begin{cases} \mathbf{u_i} + {B_i}^{-1}(\mathbf{x_j} - \mathbf{x_i}) = R_{ij}\mathbf{u_j} + \mathbf{t_{ij}} \\ \mathbf{u_j} + {B_j}^{-1}(\mathbf{x_i} - \mathbf{x_j}) = R_{ji}\mathbf{u_i} + \mathbf{t_{ji}} \end{cases}$$

REMARK 1. *This formulation does not ensure the positivity of the Jacobian determinant* $\det(J(M_i)) > 0$*, so the trivariate functions* M_i *may not provide a valid mapping. Following our definition, two maps can be grid-equivalent even if they are not valid. However in practice, most* M_i *will be valid maps at the end. Invalid ones generate singular tetrahedra (see §2.2), treated as explained in §2.3.*

2.2 Optimization

We defined a mixed-integer problem, where $\mathbf{u_i}$ are real, $\mathbf{t_{ij}}$ are integer and R_{ij} are permutation matrices. This section explains how we solve the problem. First we solve for the matrices R_{ij}. We have $J(M_i) = R_{ij}J(M_j)$ and we want $J(M_i) \approx B_i$, thus it is fair to define R_{ij} as:

$$R_{ij} = \arg\min_{R \in \mathbf{R}} \|R - {B_i}^{-1}B_j\|.$$

We can now replace $\mathbf{t_{ij}}$ by $R_{ij}\mathbf{t_{ji}}$ for $i < j$. Then we multiply the second equation by $-R_{ij}$ and regroup the terms:

$$\forall i < j, \quad \begin{cases} \mathbf{u_i} - R_{ij}\mathbf{u_j} - \mathbf{t_{ij}} & -{B_i}^{-1}(\mathbf{x_i} - \mathbf{x_j}) & = \mathbf{0} \\ \mathbf{u_i} - R_{ij}\mathbf{u_j} - \mathbf{t_{ij}} & -R_{ij}{B_j}^{-1}(\mathbf{x_i} - \mathbf{x_j}) & = \mathbf{0} \end{cases}$$

Note that if a linear system is composed of equations of type $x - a = 0$ and $x - b = 0$, then solving it *in the least squares sense* is equivalent to solving the system of equations $x - (a+b)/2 = 0$.

It leads to the new formulation:. we now have a set of equations (one per oriented edge $i < j$) to be solved in the least squares sense:

$$\forall i < j, \quad \mathbf{u_i} - R_{ij}\mathbf{u_j} - \mathbf{t_{ij}} + \mathbf{g_{ij}} = \mathbf{0} \tag{2}$$

where the variables are $\mathbf{u_i}$, $\mathbf{u_j}$ and $\mathbf{t_{ij}}$ and where the input constants $\mathbf{g}_{ij}$ are given as follows:

$$\mathbf{g_{ij}} = ((B_i^{-1} + R_{ij}B_j^{-1})/2)(\mathbf{x_j} - \mathbf{x_i}) \tag{3}$$

$$R_{ij} = \arg\min_{R \in \mathbf{R}} \|R - {B_i}^{-1}B_j\| \tag{4}$$

Recall that $\mathbf{t_{ij}}$ are integer triplets. As each $\mathbf{t_{ij}}$ appears exactly in one line of the system (2), we propose to consider the *fractional* part of each equation, it allows us to solve a system without constraints of integrality and then to recover $\mathbf{t_{ij}}$:

$$\forall i < j, \quad \mathbf{u_i} + \mathbf{g_{ij}} = R_{ij}\mathbf{u_j} \mod 1$$

We split this vector equation into six scalar ones and use the periodicity of the cosine and sine functions to remove the modulo:

$$\forall i < j, \forall d \in \{0,1,2\}, \quad \begin{cases} \cos(2\pi(\mathbf{u_i} + \mathbf{g_{ij}})[d]) & = & \cos(2\pi(R_{ij}\mathbf{u_j})[d]) \\ \sin(2\pi(\mathbf{u_i} + \mathbf{g_{ij}})[d]) & = & \sin(2\pi(R_{ij}\mathbf{u_j})[d]) \end{cases}$$

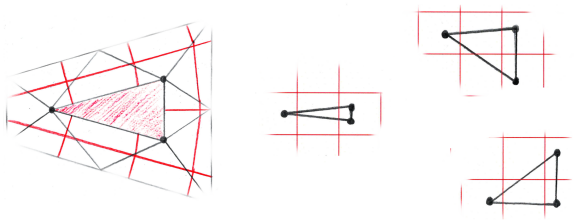


Fig. 6: A PGP-singularity corresponds to a Y-junction in the generated mesh. Left image: the Y-junction. Right images: three (non grid-equivalent) maps for the (grayed-out) singular tetrahedron.

By expanding the cosine and sine of a sum, one obtains:

$$\begin{aligned} &\forall i < j, \forall d \in \{0,1,2\}, \\ &\cos(2\pi\mathbf{u_i}[d])\cos(2\pi\mathbf{g_{ij}}[d]) - \sin(2\pi\mathbf{u_i}[d])\sin(2\pi\mathbf{g_{ij}}[d]) \\ &\qquad - \cos(2\pi(R_{ij}\mathbf{u_j})[d]) = 0 \\ &\sin(2\pi\mathbf{u_i}[d])\cos(2\pi\mathbf{g_{ij}}[d]) + \cos(2\pi\mathbf{u_i}[d])\sin(2\pi\mathbf{g_{ij}}[d]) \\ &\qquad - \sin(2\pi(R_{ij}\mathbf{u_j})[d]) = 0 \end{aligned}$$

Then we perform a change of variables. Each scalar variable $\mathbf{u_i}[d]$ is replaced with two variables $\mathbf{a_i}[d]$ and $\mathbf{b_i}[d]$ that correspond to its cosine and sine, respectively:

$$\mathbf{a_i}[d] \stackrel{\text{def}}{=} \cos(2\pi\mathbf{u_i}[d]) \qquad \mathbf{b_i}[d] \stackrel{\text{def}}{=} \sin(2\pi\mathbf{u_i}[d]).$$

Note that as $R_{ij} \in \mathbf{R}$ is one of 24 rotation matrices, for any d the corresponding row $R_{ij}[d,\cdot]$ contains two zeroes and one 1 or -1. Let us define r_{ij}^d to be the index of the non-zero entry in the row $R_{ij}[d,\cdot]$, and s_{ij}^d its sign. We can write $(R_{ij}\mathbf{u_j})[d] = R_{ij}[d,\cdot]\mathbf{u_j} = s_{ij}^d\mathbf{u_j}[r_{ij}^d]$. Then we solve the following linear system in the least squares sense:

$$\begin{aligned} &\mathbf{a_i}[d]\cos(2\pi\mathbf{g_{ij}}[d]) - \mathbf{b_i}[d]\sin(2\pi\mathbf{g_{ij}}[d]) - \mathbf{a_j}[r_{ij}^d] = 0 \\ &\mathbf{b_i}[d]\cos(2\pi\mathbf{g_{ij}}[d]) + \mathbf{a_i}[d]\sin(2\pi\mathbf{g_{ij}}[d]) - s_{ij}^d\ \mathbf{b_j}[r_{ij}^d] = 0 \end{aligned} \tag{5}$$

We force points to be generated exactly on the volume boundary, on hard edges and on corners by the fixing some variables: for each vertex v of each tetrahedron's face located on the boundary, we constrain the variables $\mathbf{a_i}[d] = 1$ and $\mathbf{b_i}[d] = 0$ if the angle between the d^{th} column of B_i and the normal (or its opposite) of the tetrahedron's face is lower than $\pi/8$. Note that most sharp features are naturally enforced by having more than two constraints for the same vertex.

To solve the linear system, we use the implementation of the Jacobi preconditioned Conjugate Gradient algorithm [Ashby et al. 1990], available in [Lévy 2000]. We then retrieve the original variables as $\mathbf{u_i}[d] \leftarrow \text{atan2}(\mathbf{b_i}[d], \mathbf{a_i}[d])$. Once we have all $\mathbf{u_i}$, the corresponding $\mathbf{t_{ij}}$ are straightforward to compute. Our resolution system does not necessarily ensure that $\mathbf{a_i}[d]^2 + \mathbf{b_i}[d]^2 = 1$ (assumed when introducing $\mathbf{a_i}$ and $\mathbf{b_i}$), but still provides satisfying results.

Singularities. As previously mentioned, our choice of variables satisfies the first two equations of the system (1). However, the last two equations can be violated. In this case, the tetrahedron is said to be *singular*. There are two types of singularities:

DEFINITION 2. *FF-Singularity (Frame-Field):*

—*The triangle* ijk *is said to be FF-singular if* $R_{ij}R_{jk}R_{ki} \neq Id_{3\times3}$*;*

—*A tetrahedron is FF-singular if any of its faces is FF-singular.*

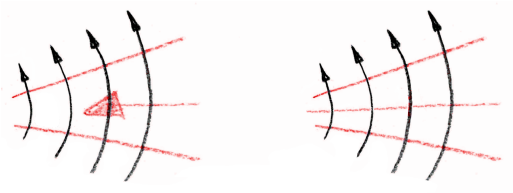


Fig. 7: Unit-norm vector fields with curl are not integrable; PGP gives a scalar field whose gradient is close to the given vector field. **Left:** integrating without cancelling the curl of the black field creates one singular triangle (highlighted in red). Red lines show integer iso-values of the scalar field, thus forming quads with the black lines (integer iso-values of the integrated orthogonal field). **Right:** integration with curl correction eliminates the singularity at the expense of introducting more distorted hexahedra.

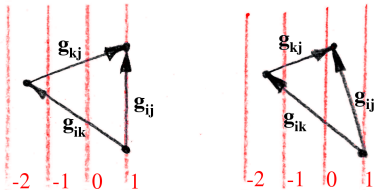


Fig. 8: Boundary constraints acting on two 1-forms $\mathbf{g}$: the **left one** respects the boundary constraint for the border edge ij, the **right one** does not (remember that the normal vector $\mathbf{n}$ becomes one of the global axes of the parametric space under the action of our maps M). Red vertical lines show integer iso-values of the first axis of the parametric domain. When we solve for our parameterization with objective $\mathbf{u_i} - R_{ij}\mathbf{u_j} - \mathbf{t_{ij}} + \mathbf{g_{ij}} = 0$ (eq. (2)) and constraints of integrality for $\mathbf{u_i}$ and $\mathbf{u_j}$, in the left case we would obtain $\mathbf{u_i}[0] = \mathbf{u_j}[0] = 0$ and in the right one $\mathbf{u_i}[0] = 0, \mathbf{u_j}[0] = 1$, thus creating hexahedra that are not aligned with the boundary.

DEFINITION 3. *PGP-Singularity (Periodic Global Param.):*

—The triangle ijk is said to be PGP-singular if $\mathbf{t_{ij}} + R_{ij}\mathbf{t_{jk}} \neq \mathbf{t_{ik}}$;
—A tetrahedron is PGP-singular if any of its faces is PGP-singular.

Note that the variables $\mathbf{u_i}$ do not appear in this criterion.

Frame-Field singularities create degenerate maps, since the FF-singularity condition enforces $\mathbf{t_{ij}} = \mathbf{0}$ and $\mathbf{u_i} = \mathbf{0}$. PGP singularities yield Y-junctions in the generated hexahedral mesh (Fig. 6).

2.3 Extracting gridpoints

Once the vectors $\mathbf{u_i}$ and $\mathbf{t_{ij}}$ are computed, it is easy to extract the gridpoints, as follows: For each tetrahedron $ijkl$ we choose an arbitrary map among M_i, M_j, M_k, M_l and then we extract the pre-images of the points with integer texture coordinates inside the image of the tetrahedron. There are two cases: either the tetrahedron is non-singular and then the result does not depend on the choice of the map, or it is singular and then the result can depend on the map. If the tetrahedron is singular, to ensure a sufficient sampling density, we select among M_i, M_j, M_k, M_l the one that maximizes the volume of the mapped tetrahedron. In general, this generates points (and then Delaunay tetrahedra) that are not recombined into hexahedra in the subsequent step.

Note that due to numerical inaccuracies there is a risk of extracting multiple copies of a gridpoint or to loose a point. It is possible to "sanitize" the parameterization as done in [Ebke et al. 2013], however, in our case, degeneracies in 3D are less likely to occur than for quads embedded in triangle meshes. Therefore, we just detect duplicated points by sorting the points spatially.

2.4 Optional pre-processing step: curl correction

Curl correction[1] is an optional step that can be used to pre-process the frame field (B_i) that steers the optimization from Section 2.2. In a nutshell, it adds a corrective term $\mathbf{c}_{ij}$ to the input $\mathbf{g}_{ij}$ as $\mathbf{g_{ij}} \leftarrow \mathbf{g_{ij}} + \mathbf{c_{ij}}$, in such a way that our optimization algorithm produces less singularities. It basically trades a higher proportion of grid-compatible maps for a larger distance to the geometric objective $J(M_i) = B_i^{-1}$. Figure 7 provides an illustration.

We first define the constraints on $\mathbf{c_{ij}}$ that would enforce grid-compatible maps §2.4.1, then we derive an algorithm to compute values of $\mathbf{c_{ij}}$ that limit both the distortion §2.4.2 and the degeneracies.

[1]In DEC language, if $R_{ij} = Id_{3\times3}$, this step is a modification of the trivariate 1-form $\mathbf{g}$ to make it closed, i.e. canceling its curl.

2.4.1 *Curl-correction constraints on $\mathbf{c_{ij}}$.* Two maps M_i and M_j are grid compatible iff each tetrahedron incident to the vertices i and j satisfies Equation 1. This means that for each triangle ijk, we have $M_i(\mathbf{x_k}) = R_{ij}M_j(\mathbf{x_k}) + \mathbf{t_{ij}}$. We can express it with our set of variables $\mathbf{u_i}$ as:

$$R_{ik}\mathbf{u_k} - R_{ij}R_{jk}\mathbf{u_k} = R_{ij}\mathbf{t_{jk}} + \mathbf{t_{ij}} - \mathbf{t_{ik}}$$

Assuming that the geometric objective is perfectly satisfied (Equation 2), we can write:

$$\begin{aligned} R_{ik}\mathbf{u_k} - R_{ij}R_{jk}\mathbf{u_k} &= R_{ij}(\mathbf{u_j} - R_{jk}\mathbf{u_k} + \mathbf{g_{jk}}) && (6)\\ &+ \mathbf{u_i} - R_{ij}\mathbf{u_j} + \mathbf{g_{ij}} && (7)\\ &- (\mathbf{u_i} - R_{jk}\mathbf{u_k} + \mathbf{g_{ik}}) && (8) \end{aligned}$$

As we are interested only by triangles ijk that are not a singularity of the frame field, we have $R_{ij}R_{jk}R_{ki} = Id_{3\times3}$ so the condition simplifies to: $R_{ij}\mathbf{g_{jk}} + \mathbf{g_{ij}} - \mathbf{g_{ik}} = \mathbf{0}$. As a consequence, we can assert the following:

REMARK 2. *If the $\mathbf{c_{ij}}$'s are such that $R_{ij}(\mathbf{g_{jk}} + \mathbf{c_{jk}}) + (\mathbf{g_{ij}} + \mathbf{c_{ij}}) - (\mathbf{g_{ik}} + \mathbf{c_{ik}}) = \mathbf{0}$, then there is a trivial solution to our optimization process (without boundary condition), with zero energy, and that produces only grid compatible maps.*

It gives us a simple sufficient condition on the $\mathbf{c_{ij}}$'s to produce grid-equivalent maps. To have hexahedra faces located on the surface boundary, the image of boundary faces and edges of T are constrained to live in an iso-value of a coordinate of the map. This boundary condition is expressed as the following set of constraints:

$$(\mathbf{g_{ij}} + \mathbf{c_{ij}}) \cdot ((B_i^{-1} + R_{ij}B_j^{-1})\mathbf{n}) = 0,$$

where $\mathbf{n}$ is the normal of the surface on edge ij located on the boundary of the surface. Figure 8 shows an illustration for this constraint. If it is not respected, then corrected boundary edge $\mathbf{g_{ij}} \leftarrow \mathbf{g_{ij}} + \mathbf{c_{ij}}$ may be not aligned with the global axes of the parametric domain, creating hexahedra that do not conform with the boundary.

2.4.2 *Curl-Correction Algorithm.* We first search for the corrective term $\mathbf{c_{ij}}$ of minimal norm that respects the constraints, in other words we solve the following constrained optimization problem:

$$\min \Sigma_{ij} \|\mathbf{c_{ij}}\|^2$$

$$\text{s.t.} \begin{cases} \text{For all non-singular triangles } ijk, \\ \quad R_{ij}(\mathbf{g_{jk}} + \mathbf{c_{jk}}) + (\mathbf{g_{ij}} + \mathbf{c_{ij}}) - (\mathbf{g_{ik}} + \mathbf{c_{ik}}) = \mathbf{0} \\ \text{For all edges on the border } ij, \\ \quad (\mathbf{g_{ij}} + \mathbf{c_{ij}}) \cdot ((B_i^{-1} + R_{ij} B_j^{-1})\mathbf{n}) = 0 \end{cases}$$

The resulting $\mathbf{c_{ij}}$ ensures that Equation 1 is respected everywhere. However, it can produce highly distorted (and probably not injective) maps. To avoid this, we scale the result by:

$$\mathbf{c_{ij}}[d] \leftarrow \min\left(1, \frac{\gamma \|\mathbf{g_{ij}}\|}{\|\mathbf{c_{ij}}\|}\right) \mathbf{c_{ij}}[d]$$

The parameter γ sets the minimum ratio between the corrected desired scale and the original one. The impact of γ is discussed in §4.2.1, and all other results are obtained with the default setting $\gamma \leftarrow 0.35$.

3. GENERATING THE HEXAHEDRAL-DOMINANT MESH

The algorithm described in the previous section §2.3 generates a set of points P. The points in P are well spaced and mostly organized on a warped regular grid (except on the singularities). In addition, P samples the border ∂T of the input tetrahedral mesh and fills its interior. Our method to generate a hexahedral-dominant mesh from P can be summarized as follows (more details further):

—**Step 1:** remesh of the border of the domain ∂T using as vertices the points of P that are located on the border of T (§3.1);

—**Step 2:** compute an intermediate tetrahedral mesh T' using the points P constrained by the remeshed border. One can use an off-the-shelf constrained Delaunay implementation, such as MGTetra [George et al. 1990] or tetgen [Si 2015];

—**Step 3:** recombine in T' the sets of tetrahedra that can be assembled to form a hexahedron. Optionally, if supported by the application that uses the mesh, prisms and pyramids can be generated as well. We use a refinement of the algorithm proposed by Meshkat and Talmor in [2000] §3.2.

3.1 Remeshing the border of the domain

This section explains how to remesh the border ∂T of the input tetrahedral mesh T to generate a new triangulated surface with its vertices in the pointset P generated at the previous step (plus some additional vertices, as explained later). We start by computing $\text{Del}(P)|_{\partial T}$, the Delaunay triangulation of P restricted to the border of the domain ∂T. In other words, this means computing the intersection between the Voronoi diagram of P and ∂T (thin yellow lines in Fig. 9 top-left). Each time three Voronoi cells meet (i.e. when a Voronoi edge has a non-empty intersection with ∂T), then the corresponding three points are connected with a triangle (bottom-left). We use the algorithm described in [Yan et al. 2009] implemented in [Lévy 2015] with arithmetic filters [Meyer and Pion 2008], exact arithmetics using expansions [Shewchuk 1997] and symbolic perturbations [Edelsbrunner and Mücke 1990]. Such perturbations are useful to disambiguate triangle connections whenever four Voronoi cells meet (instead of three in the generic case). In our specific case, such degenerate configurations occur very often since the points that we generate are aligned on a regular grid.

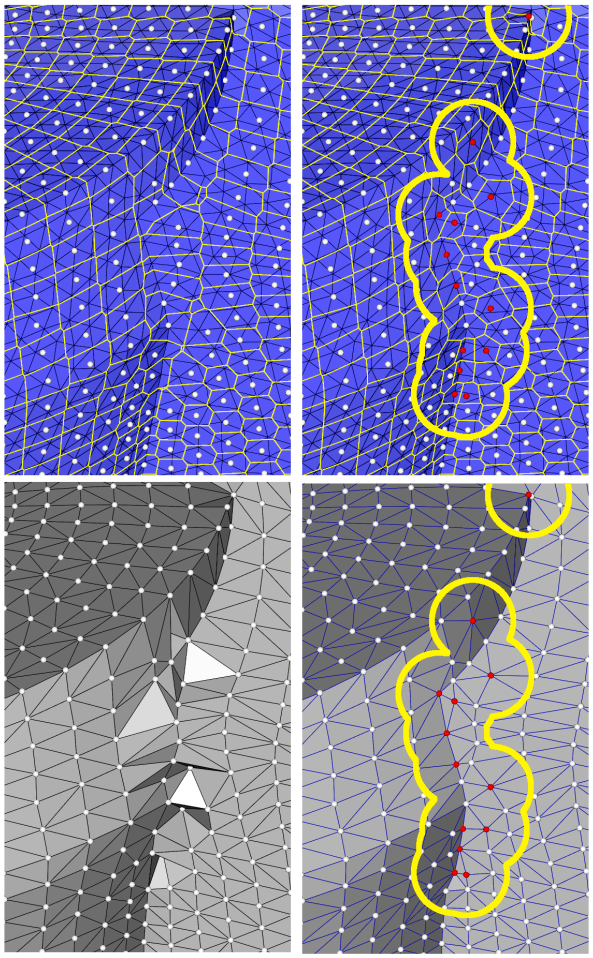


Fig. 9: The border is remeshed using the Delaunay triangulation of the generated points (white) restricted to the border of the domain. Additional points (red) are inserted until the geometric error is smaller than a user-defined threshold.

Note that the sampling of ∂T realized by P is not always sufficiently dense to capture all the geometric features of ∂T (see Fig. 9 bottom-left). For this reason, we iteratively insert points in the regions of largest geometric error (red points on Fig. 9) until the Hausdorff distance between ∂T and the remeshed border is smaller than a user-defined threshold ε. The refinement algorithm is detailed in Algorithm 1 (next page).

The algorithm iteratively adds batches of points B. To facilitate reproducing our results, we further detail some parts of the algorithm. Line 2: the 'sample_surface' function generates a regular sampling for each triangle of ∂T in such a way that the maximal distance between two samples is smaller than ε. Line 6: the function 'distance_to_surface' gives the minimal distance between a point and a triangulated surface, implemented using an AABB-tree. Line 11: the constant ϵ' specifies the minimal distance between points inserted in the same batch B (see condition Line 13). We use $\epsilon' = 2 \times \text{ave\_edge\_length}(T)$ where 'ave_edge_length' denotes the average edge length of the input mesh. We found this value empirically: a lower value generates useless points, and a higher value makes the algorithm slower by inserting smaller batches and thus

1 RefinePointSet(T,P,ε,ε'):

Data: T: the input tetrahedral mesh, and ∂T its boundary; P: the pointset to be refined; $\varepsilon > 0$: the maximum distance between the remeshed boundary and the input boundary ∂T;
Result: The updated pointset P with new inserted point to make the distance between Del(P)$|_{\partial T}$ and ∂T smaller than ϵ.

```
2  pointset E ← sample_surface(∂T, ε)
3  pointset B ← ∅
4  do
5      for i := 1 to |E| do
6          d[i] ← distance_to_surface(E[i], Del(P)|∂T)
7      end
8      sort (E, d) by decreasing d
9      B ← ∅  ;  i ← 0
10     // Min. dist. between two points inserted in the same batch
11     constant ε' ← 2 × avg_edge_length(T)
12     while i < |E| and d[i] < ε/2 do
13         if distance_to_pointset(E[i], B) > ε' then
14             B ← B ∪ {E[i]}
15         end
16         i ← i + 1
17     end
18     P ← P ∪ B
19 while B ≠ ∅;
```

Algorithm 1: The mesh refinement algorithm

requiring a larger number of outer iterations. Line 13: the function 'distance_to_pointset' gives the minimal distance between the input point and all the elements of the pointset. The parameter ε ensures that all the points of ∂T are at most at a distance of ε of the remeshed border. We use $\varepsilon \leftarrow 0.2 \times$ ave_edge_length(T). The effect of this parameter is discussed in §4.2.

3.2 Recombining tetrahedra into hexahedra

Once the border is remeshed (by Del$(P)|_{\partial T}$), we can obtain an intermediate tetrahedral mesh T' by computing the Delaunay triangulation of the pointset P constrained by the remeshed border, using off-the-shelf software [George et al. 1990; Si 2015]. In this section, we explain how to deduce from this intermediate tetrahedral mesh a hexahedral-dominant mesh by recombining the tetrahedra into other primitives (hexahedra, and optionally prisms and pyramids). We first present the general problem statement:

3.2.1 *Optimal Recombination: Problem statement.*

Given the following elements (see Fig. 10):

—an intermediate tetrahedral mesh $T' = \{t_i\}_{i=1}^{nt}$ where t_i denotes one of the tetrahedra and nt the number of tetrahedra;

—the set of "primitive templates" $\mathbf{G}$ to be recognized in T' (hexahedra, prisms, pyramids). Each template $G \in \mathbf{G}$ is a graph that encodes the combinatorial relations within a set of tetrahedra that corresponds to a primitive. This combinatorial representation comprises the adjacencies of each pair of tetrahedra along their common facets and additional information (more on this below). It is said that a primitive represented as a set of tetrahedra $H = (t_1, t_2, \dots t_k)$ *matches* a template G if the adjacencies between $(t_1, t_2, \dots t_k)$ correspond to the arcs of the graph G (a more formal definition will be given later).

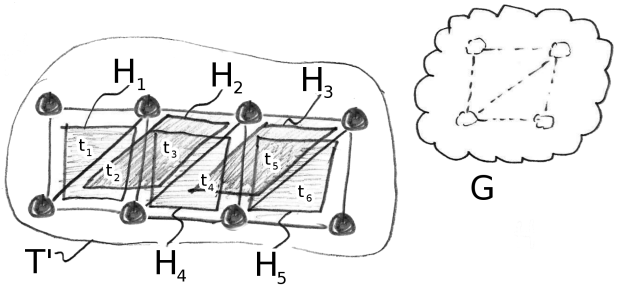


Fig. 10: Recombination (in 2D): the template G describes the recombination of two triangles into a quad. Five instances $\mathbf{H}^* = \{H_1, H_2, H_3, H_4, H_5\}$ can be found in the intermediate mesh T'. Clearly, H_2 and H_3 are of lower quality (lower value of function Q). H_1 and H_2 are mutually incompatible ($C(H_1, H_2) = 1$) because they have t_2 in common. The other mutually incompatible pairs are (H_2, H_4), (H_4, H_3) and (H_3, H_5).

—a criterion $Q(H) > 0$ that measures the geometric quality of a recognized primitive H. The criterion Q that we use is explicited further (see Section 3.2.4);

—a set of compatibility constraints $C(H_1, H_2)$. If both primitives of the pair can be constructed simultaneously in the resulting mesh, then they are said to be *mutually compatible* ($C(H_1, H_2) = 0$), otherwise they are *mutually incompatible* ($C(H_1, H_2) = 1$). Clearly, two primitives H_1 and H_2 that use the same tetrahedron are mutually incompatible ($H_1 \cap H_2 \neq \emptyset \Rightarrow C(H_1, H_2) = 1$). There are also less trivial compatibility constraints, described below (Section 3.2.4).

the optimal recombination problem can be stated as:

> Find the set of primitives $\mathbf{H}$ that maximizes the quality $Q(\mathbf{H}) = \sum_i Q(H_i)$, such that each primitive $P \in \mathbf{H}$ matches one of the templates $G \in \mathbf{G}$ and such that the mutual compatibility constraints are satisfied ($\forall H_1 \neq H_2 \in \mathbf{H}, C(H_1, H_2) = 0$).

3.2.2 *Re-formulation / decomposition.* This combinatorial optimization problem can be decomposed into two steps:

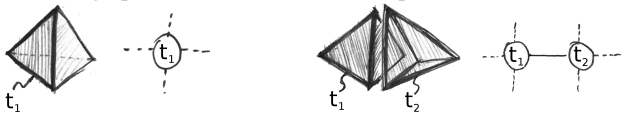
> *Step 1 - Template matching:*
> Find the set of all possible primitives $\mathbf{H}^*$ extracted from the intermediate tetrahedral mesh T' that match a template in $\mathbf{G}$.
>
> *Step 2 - Constrained optimization:*
> $\text{Max}_{\mathbf{b}\in\{0,1\}^N} \left[\sum_{i=1}^{N} b_i Q(H_i)\right]$ where $N = |\mathbf{H}^*|$
> s.t. $\forall i, j \in (1..N)^2, b_i b_j C(H_i, H_j) = 0$

Template matching (Step 1) is detailed further, in Section 3.2.3. In the constrained optimization problem of Step 2, one tries to find the *boolean vector* $\mathbf{b}$ that indicates for each potential primitive $H_i \in \mathbf{H}^*$ whether it will be used ($b_i = 1$) or not ($b_i = 0$) in the final result. The constraint indicates that whenever a primitive H_i is used, the primitives H_j that are incompatible with it cannot be used. Written in this form, this (combinatorial) constrained optimization problem can be recognized as the *maximum independent set problem*, classical in graph theory [Bomze et al. 1999]. Since we will be able to find the set of all candidate primitives $\mathbf{H}^*$, this lets hope for an algorithm that provably finds the optimum recombination. Unfortunately, even for a few hundred elements,

algorithms for the maximum independent set problem take a considerable amount of time, as confirmed by our experiments. Therefore, for this step, we use a classical greedy heuristic (more on this in Section 3.2.4).

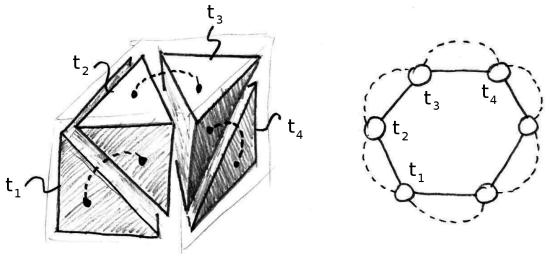
We now give more details about each phase of the algorithm, template matching (Section 3.2.3) and combinatorial optimization (Section 3.2.4).

3.2.3 *Template matching.* Several algorithms were proposed for recombining tetrahedra into hexahedra [Yamakawa and Shimada 2003; Meshkat and Talmor 2000]. We chose to elaborate on the approach proposed by Meshkat and Talmor [2000], that provides a formalism that can be used to systematically analyze all the configurations. Their formalism represents a configuration as a graph, where each node corresponds to a tetrahedron:



Each (solid) arc connects two tetrahedra that share a facet. The dangling dashed arcs correspond to facets on the border, not connected to another tetrahedron.

In addition, to identify the quadrilateral facets in the decomposition of a hexahedron, they connect each pair of dangling dashed edges that correspond to the pair of triangular facets that form each quadrilateral facet:



In the example shown above, three dashed arcs are materialized on the left image (the three other hidden ones are in a similar configuration).

Meshkat and Talmor refer to such a graph (with both solid and dashed arcs) as the *augmented graph* of the decomposition. They enumerate the possible augmented graphs for decompositions with 5 and 6 tetrahedra. They mention (without describing it) a possible generalization in the presence of slivers. In Appendix A, we further analyze their formalization, fill a gap in the proof, prove that a configuration cannot contain more than 13 tetrahedra, extend the analysis to all the possible configurations (from 5 to 13 tetrahedra), summarized in the following theorem:

THEOREM 4. *A decomposition of a hexahedron into tetrahedra without any sliver on the border can have 5, 6 or 7 tetrahedra. There is 1 configuration with 5 tetrahedra, 5 configurations with 6 tetrahedra and 4 configurations with 7 tetrahedra (see Fig. 11)*

PROOF. See Appendix A □

In addition, a sliver can be "glued" to each quadrilateral face of the hexahedron:

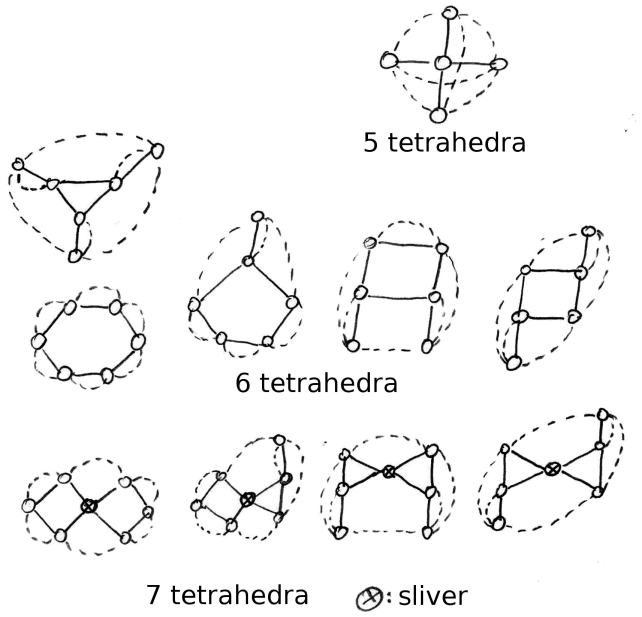


Fig. 11: The augmented graphs of the ten decompositions of a hexahedron into tetrahedra.

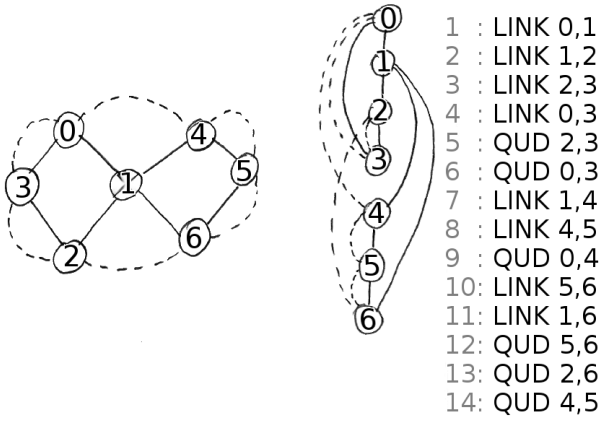
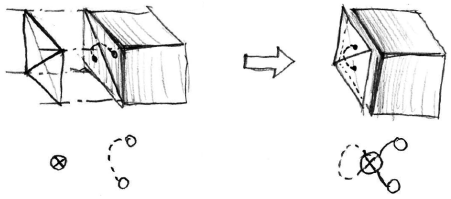


Fig. 12: An augmented graph (left) is linearized (center) and transformed into a program (right).



where the sliver is symbolized by a circled cross. The corresponding graph transform can be applied to each dashed arc, thus generating up to 2^6 graphs from each initial graph (modulo symmetries). Thus, a decomposition can have up to 13 tetrahedra (the maximum 13 is reached with one of the decompositions into 7 tetrahedra with 6 slivers glued on the quadrilateral facets).

Now that all the possible configurations of a hexahedron are known, we can proceed to describe the algorithm that recognizes them in the intermediate tetrahedral mesh T'. This is an instance of the subgraph isomorphism problem, known to be NP-complete [Cook 1971]. However, since the template graph to be recognized is small (no more than 7 nodes), systematic exploration with backtracking remains reasonably efficient. We follow the approach in [Meshkat and Talmor 2000], that first transforms each augmented graph into a "program", as exemplified in Fig. 12. Each node (tetrahedron) of the graph is numbered (left), then the graph is "linearized" (center). The "program-form" of the graph is a sequence of `LINK` and `QUAD` instructions that encode the solid and dashed arcs respectively.

At each step of the program, nodes touched by a previous `LINK` instruction are said to be *visited*. Node 0 is initially considered to be visited as well, before the program starts. To facilitate the design of the matching algorithm described below, the `LINK` and `QUAD` instructions are scheduled in the program in such a way that:

—a `QUAD` instruction always connects two already visited nodes;

—a `LINK` instruction can connect an already visited node to a new one (the new one is then visited), as in lines 1,2,7,8,10;

—or a `LINK` instruction can bridge two already visited nodes (lines 4 and 11).

Once each of the 10 augmented graphs of Fig. 11 is encoded as a program, all the possible primitives in the input tetrahedral mesh T' can be recognized by a simple backtracking algorithm. To ease reproducing our results, we have given in Appendix B the complete description of this algorithm.

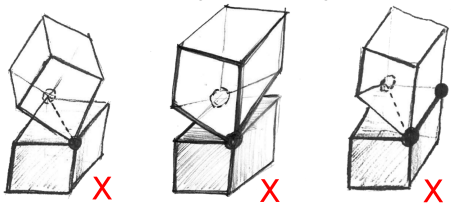
3.2.4 *Constrained optimization.* Once the template matching algorithm of the previous section is executed, we obtain the set $\mathbf{H}^*$ of all possible primitives that can be recombined in the intermediate mesh T' by applying Algorithm 3 to the 10 programs that correspond to the 10 possible decompositions of a hexahedron (and optionally to the program that recognizes prisms and the one that recognizes pyramids).

The goal now is to find the subset $\mathbf{H} \subset \mathbf{H}^*$ that maximizes the quality criterion $Q(\mathbf{H}) = \sum_{H \in \mathbf{H}} Q(H)$ and that satisfies the compatibility constraints $\forall H_i, H_j \in \mathbf{H}, C(H_i, H_j) = 0$. We use the following quality criterion that favors flat quadrilateral facets with right angles:

$$Q(H) = \sum_{\text{q} \in \text{quads}(H)} \left(\widehat{n_1, n_2}^2 + \frac{1}{4} \sum_c \left(\hat{c} - \frac{\pi}{2} \right)^2 \right)$$

where $\widehat{n_1, n_2}$ denotes the angle between the two normals n_1, n_2 of the two triangular facets recombined in the quadrilateral facet q, and where $\hat{c}$ denotes the angle at the corner c of the quadrilateral facet q.

The compatibility criterion $C(H_1, H_2)$ is defined in function of the envisioned application for the generated hexahedral-dominant mesh. In our case, the Finite Element Modeling application that we target tolerates a single type of nonconformity in the mesh, that is a quadrilateral facet connected to two triangular facets. All the other nonconformities that involve the diagonal edges of the quadrilateral faces are forbidden. In other words, this means that the following three configurations are forbidden:

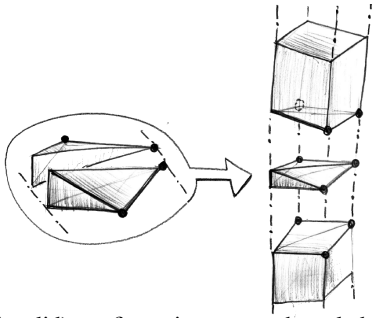


More formally, two primitives H_1 and H_2 are mutually incompatible ($C(H_1, H_2) = 1$) if one of the following conditions is met:

(1) H_1 and H_2 have one or more tetrahedra in common ($H_1 \cap H_2 \neq \emptyset$);
(2) an edge of H_1 corresponds to a diagonal edge of H_2 or vice-versa (left figure);
(3) two quadrilateral facets of H_1 and H_2 have two diagonally opposite vertices in common (center figure);
(4) two quadrilateral facets of H_1 and H_2 have three vertices in common (right figure).

Otherwise, H_1 and H_2 are mutually compatible ($C(H_1, H_2) = 0$).

REMARK 3. *At first sight, one may think that the invalid configuration (4) can be avoided by simply ensuring that the pair of tetrahedral facets adjacent to a quadrangular facet are connected to the same hexahedron. However, there exists a configuration where two hexahedra share three vertices without any direct adjacency between their tetrahedra:*



The shown (invalid) configuration cannot be ruled-out by simply examining tetrahedron facet adjacencies. As a consequence, the algorithm needs to traverse the set of cells incident to each vertex.

Since there is no efficient algorithm that finds the exact set of primitives that maximizes Q while satisfying the compatibility constraints (maximum independent set problem), we use instead the heuristic outlined in Algorithm 2.

Implementation details / Optimizations: To avoid the n^2 cost of testing mutual compatibility for all primitive in $\mathbf{H}^*$ against all the already recognized ones in $\mathbf{H}$, we observe that incompatibility between two primitives only occur when they share at least a vertex. Thus we restrict the compatibility test to the set of primitives that share a vertex with H_1.

Occurence of the configurations with slivers: As shown in the statistics below (obtained for the *fusee* model displayed next page), the additional configurations with an internal sliver X appear in practice (second row). Though they are less often encountered than the regular one (first row), detecting them reconstructs a non-negligible number of additional hexahedra (all the configurations are depicted in Figure 26 of the appendix).

SelectPrimitives($\mathbf{H}^*, T'$):

Data: $\mathbf{H}^*$: the set of all the possible primitives recognized in T'

Result: $\mathbf{H}$: a set of mutually compatible primitives selected from $\mathbf{H}^*$

```
sort H* by primitive type: hexahedron > prism > pyramid
sort primitives of same type by decreasing Q(H)
H ← ∅
foreach H1 ∈ H* do
    if (∀H2 ∈ H, C(H1, H2) = 0) then
        H ← H ∪ {H1}
    end
end
```

Algorithm 2: The constrained optimization algorithm

$G_I - G_I$ 601270	$G_I - G_{II}$ 81907	$G^+_{II} - G^+_{II}$ 181616	$G^+_{II} - G^-_{II}$ 40358
$G_I - X - G_I$ 674	$G_I - X - G_{II}$ 101	$G^+_{II} - X - G^+_{II}$ 5846	$G^+_{II} - X - G^-_{II}$ 2252

4. RESULTS AND DISCUSSION

We evaluate the quality of our results by the proportion of hexahedra elements, and some measures of their geometric quality. Our algorithm was tested on a large set of models (available in supplemental material) and is discussed for a representative subset in this section.

We compare the classic hexahedral mesh quality measures with previous works in §4.1, discuss the parameters and their influence in §4.2, and we discuss its strengths and weaknesses in term of robustness.

4.1 Hexahedra proportion and quality

We tested our method on various modeled shapes and CAD models with a 2.2Ghz Intel Core i7 CPU and 8GB of RAM laptop. Table I summarizes the resulting statistics and timings. We copied the results reported in [Carrier-Baudouin et al. 2014] in Table I (lines in gray) for comparison purposes. In all our experiments we made the number of vertices generated by our method as close as possible to the other method for comparison purposes. To make the comparison accurate, we used exactly the same criteria as in [Carrier-Baudouin et al. 2014], i.e. the scaled Jacobian is only measured on hexahedral elements. Note that it would be also possible to measure the scaled Jacobian on the tetrahedra elements, but it would not be pertinent, since the reference tetrahedron (with vertices $(0,0,0), (1,0,0), (0,1,0), (0,0,1)$) does not have a regular shape, thus measuring deformations relative to it would be completely biased (it is probably why Carrier-Baudoin *et al.* do not measure the scaled Jacobian on tetrahedra either).

The resulting meshes are displayed on top of Table I. The recombination algorithm is configured to generate hexahedra (in white), prisms (in green) and pyramids (in blue). Prisms and pyramids are generated by detecting the (unique) combinatorial configurations of three (respectively two) tetrahedra that correspond to them. The tetrahedra that were not recombined are shown in red. Recombining prisms and pyramids is optional, and targeted to finite element softwares that can use them.

Examples of extra points addition can be observed on fandisk, fertility, rockerarm and CV745 models. For these models the sampling of ∂T realized by P is not sufficient as measured by the Hausdorff distance and reported in the column $dist1$. The added points imply the creation of tetrahedra, prisms and pyramids that can be observed in several regions of these models. The impact of this step and the associated parameter is discussed below, in §4.2.2.

We observe in Table I that the percentage of hexahedra is higher in number and volume with our method despite the addition of points (e.g. CV745 model). The quality Q of the produced hexahedra is also better with our method. The significant amount of time of the refinement step is due to the number of batches required to reach the user defined threshold. Indeed, each batch of points inserted in P implies to update Del(P)$|_{\partial T}$, the Delaunay triangulation of P restricted to the border of the domain ∂T.

Comparison data for L_p-Centroidal Voronoi Tesselation [Lévy and Liu 2010] is displayed in Figure 13. As can be seen, L_p-CVT generates a larger number of non-hexahedral elements everywhere in the mesh, probably because the numerical solver gets stuck in a local minimum of the non-linear objective function. Note also the shorter computation time of our algorithm.

We also include statistics of our method applied to a larger database of models, shown at the end of the article (Table II), that comprises industrial parts of varying complexity and

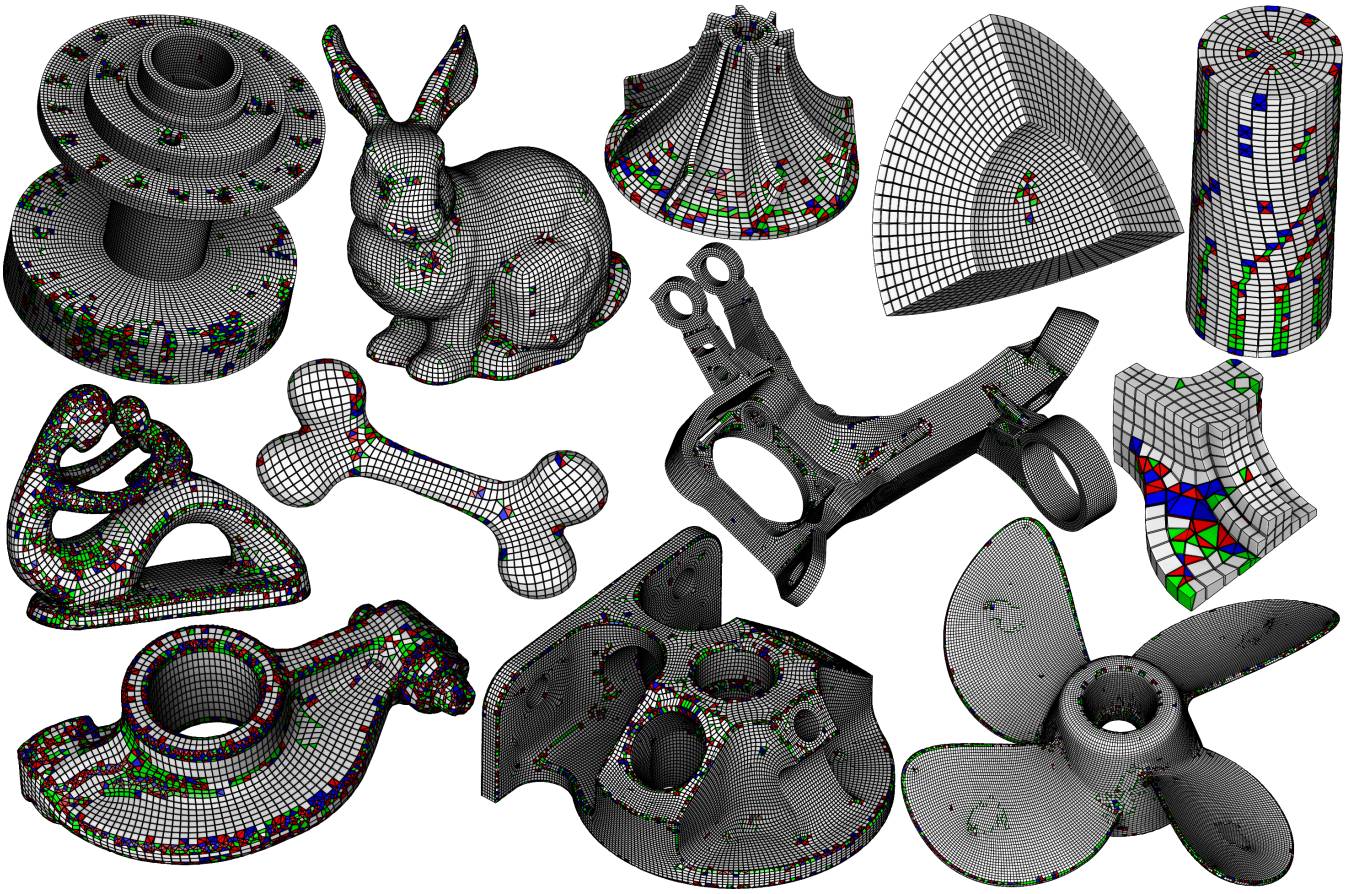
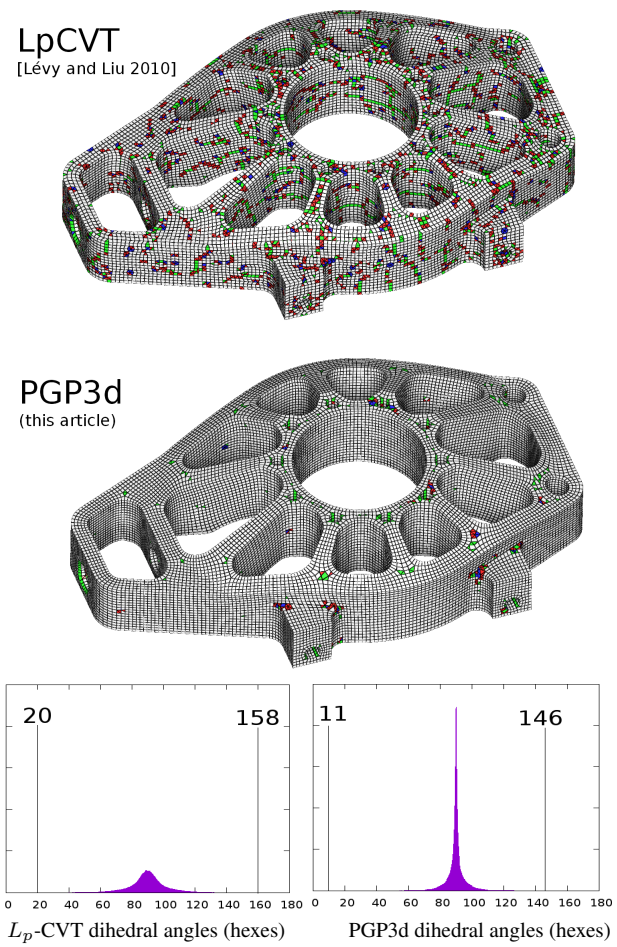


Table I. : Statistics and timings for the models above (first line: cubo, bunny, impeller, corner, cylinder; second line: fertility, bone, fusee, fandisk; third line: rockerarm, CV745, propeller). The hex. proportion is measured both in number (H_{nbr}) and volume (H_{vol}). The hex. mesh quality Q is measured by the scaled Jacobian in the format of average | standard deviation. The Hausdorff distance between our mesh and the reference input mesh is in the format of $M \rightarrow \partial T | \partial T \rightarrow M$ with M the mesh before (dist1) and after (dist2) the remeshing step. We also give the timings for the initial tetrahedralization, frame field generation (FF), curl correction (CC), global parameterization (PGP), pointset extraction, refinement step, tet. to hex. step and total time in seconds.

model	#vert	H_{nbr}	H_{vol}	Q	dist1	dist2	tet input	FF	CC	PGP	pointset	refine	tet2hex	total
fusee	131,508	81.22	94.62	0.98 | 0.04	0.387 | 0.252	0.070 | 0.060	52.41	104.52	195.87	58.76	29.28	110.82	116.42	728.32
fusee	126,922	62.91	85.70	0.95 n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	488
CV745	133,331	72.45	91.63	0.97 | 0.05	0.845 | 0.540	0.101 | 0.091	109.29	221.86	494.2	117.51	59.46	266.07	125.67	1496.42
CV745	133,436	n/a	89.74	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	247
propeller	132,469	71.27	91.10	0.97 | 0.05	0.355 | 0.327	0.035 | 0.062	47.95	127.08	191.78	51.24	28.99	119.82	123.28	748.57
propeller	133,678	n/a	83.65	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	268
cubo	109,182	72.87	89.01	0.98 | 0.04	0.782 | 0.671	0.705 | 0.257	7.45	13.22	19.5	5.14	7.74	69.54	98.01	247.17
cubo	102,946	n/a	78.55	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	225
cylinder	11,205	64.66	90.85	0.96 | 0.06	0.208 | 0.214	0.134 | 0.137	33.32	65.56	116.86	39.00	17.37	21.55	12.08	327.13
cylinder	11,648	58.16	82.68	0.94 n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	112
corner	6,538	95.46	99.55	0.99 | 0.02	0.095 | 0.102	0.095 | 0.102	2.73	5.59	6.05	2.37	1.66	0.83	6.04	27.98
corner	6,006	84.54	94.10	0.96 n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	80
fandisk	856	51.36	77.82	0.95 | 0.06	2.184 | 3.289	0.458 | 0.493	0.96	1.87	2.52	0.69	0.45	1.59	0.53	9.6
rockerarm	15,564	33.05	80.15	0.95 | 0.09	1.308 | 1.258	0.072 | 0.074	91.45	215.2	546.68	103.00	39.99	147.66	11.56	1209.32
impeller	13,896	53.31	81.11	0.95 | 0.06	1.408 | 0.848	0.333 | 0.219	4.51	8.02	11.28	3.16	2.69	11.9	10.7	59.37
bunny	116,149	60.57	88.61	0.95 | 0.06	0.791 | 0.779	0.123 | 0.129	23.74	56.1	127.88	29.62	19.66	42.6	136.43	468.89
bone	4,225	45.17	82.54	0.92 | 0.10	1.838 | 1.145	0.264 | 0.257	2.38	5.13	12.48	3.15	1.82	3.11	4.15	35.07
fertility	20,068	33.63	78.40	0.93 | 0.11	1.292 | 0.904	0.069 | 0.071	89.42	205.58	454.71	106.06	43.15	145.71	16.51	1120.7

scanned meshes, from Drexel University, GRABCAD and AimAtShape repository. Some of the results are shown in Fig. 23. The automatically-generated output (except the brain that constitutes confidential patient specific data) for the complete database is included in the supplemental material (statistics and images in a PDF file, and all the output mesh files). To experiment how the method



L_p-CVT dihedral angles (hexes) PGP3d dihedral angles (hexes)

algo	#Vert	#Hex	#Tet	#Prsm	#Pyr	H_{nb} (%)	H_{vol} (%)	time (s)
Lp-CVT	74997	45980	50475	7828	3975	42.5	72.5	1255
PGP3d	89611	69147	5599	2386	573	89	95	476

Fig. 13: Comparison of the results obtained with L_p-CVT and our method. The histograms record the repartition of dihedral angles in the generated hexahedra (with the minimum and maximum angles indicated). The table indicates the number of mesh elements (vertices, hexahedra, tetrahedra, prisms, pyramids), the proportion of hexahedra in terms of number of elements and volume, and the computation time. The same guiding direction field was used for both methods.

scales up both in terms of input complexity and number of generated cells, we used it to generate a hexahedral-dominant mesh of a complete engine block (TRX engine from GRABCAD), shown in Fig. 24.

4.2 Influence of the Parameters

All our results in the previous subsection were produced using the default parameters. We now discuss the influence of each parameter and its impact on the result quality, namely the maximal propertion of curl correction §4.2.1, the maximal deviation to the data that triggers the insertion of additional points §4.2.2 and the influence of a varying prescribed element size §4.2.3.

4.2.1 *Maximal proportion of curl correction.* The parameter γ (introduced in § 2.4.2) that tunes how much curl correction is allowed really impacts complex models (Fig. 14). High values of γ (left) make the correction term meaningless, so the algorithm introduces many Y-junctions to balance the curl of the frame field. When γ is low (right), the correction term makes the field locally integrable everywhere, exactly as in CubeCover before introducing the constraint to have integer variables. As a consequence, when γ is too low, our algorithm have the same failure cases as CubeCover.

4.2.2 *Extra points to fit the original model.* Our algorithm adds extra points to the pointset P to ensure that the hexahedral-dominant mesh is close enough to the input mesh §3.1. It stops when it does not find any point of the input mesh that is further than a given threshold to the current reconstructed tetrahedral mesh. The impact of this threshold is illustrated in Fig. 15: if the threshold is high, our algorithm fails to capture details of the model, but if it is too low, then it adds too many extra points to curved areas and thus prevents some hexahedra from being reconstructed.

4.2.3 *Varying scale.* In our method, the scale of the hexahedra is prescribed by the norm of the columns of B_i. We demonstrate a varying isotropic scaling on a real object Fig. 16–up, and an anisotropic scaling on a cube Fig. 16–bottom. Our method handles the scale variation by introducing new singularities that split a layer of cubes into two layers of cubes. Intuitively, this behavior may be considered as the $3D$ counterpart of the T-vertices used in quadrilateral surface remeshing.

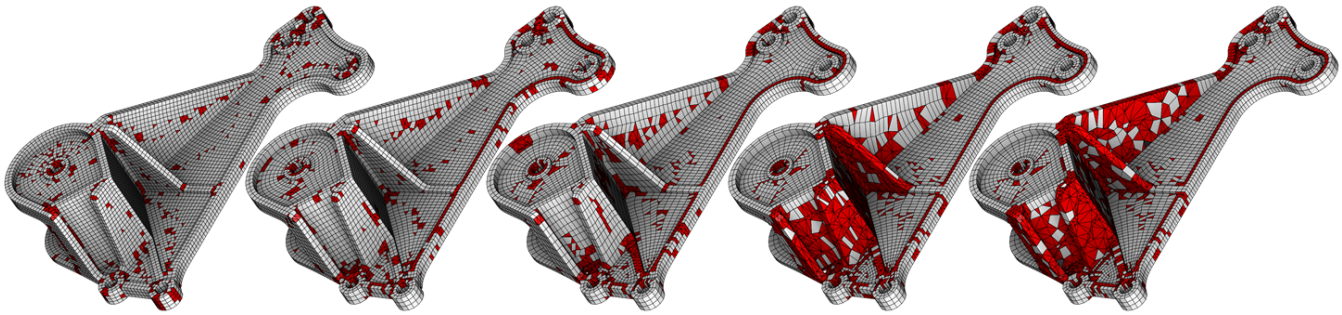


Fig. 14: Influence of the maximum curl-correction parameter $\gamma \in \{1, 0.8, 0.6, 0.4, 0.2, 0\}$. A value of γ that is too low results in the same over-constrained problem as in CubeCover (right images).

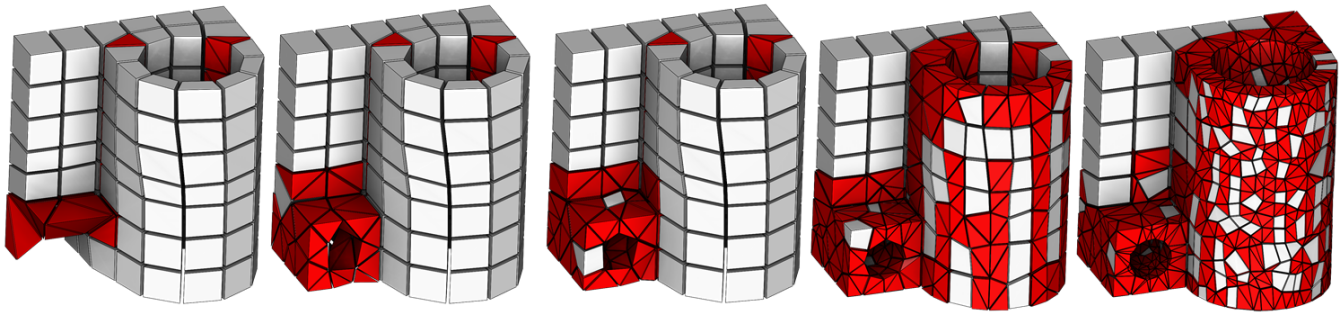


Fig. 15: Influence of the maximal deviation parameter $\epsilon \in \{100\%, 2\%, 1\%, 0.5\%, 0.25\%\}$ of the average edge length of the input mesh. A high tolerance misses some features (left), whereas a too strict tolerance inserts too many points (right).

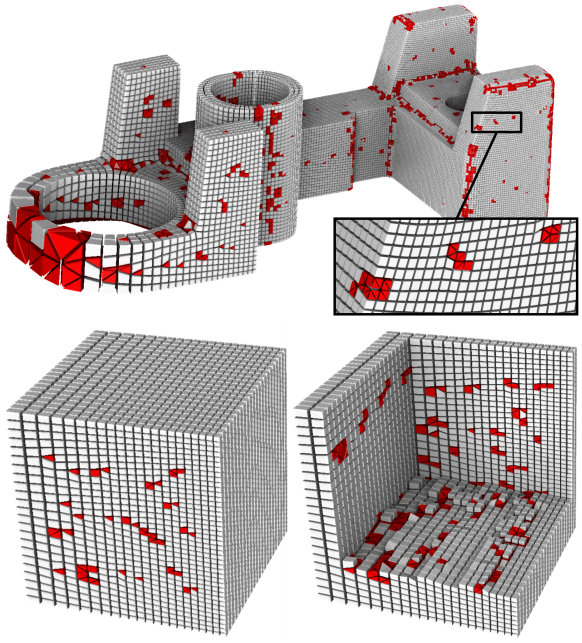


Fig. 16: Varying scale: In the upper image, the desired hexahedra size varies linearly with the x coordinate. The bottom row shows a cube (clipped on the right image) with a varying anisotropic scale. Singularities are evenly distributed to absorb the variation of resolution.

4.3 Robustness

Recent advances in pure hexahedral remeshing [Li et al. 2012] produce very good results, but are limited by global constraints due to the topology of the input frame field. We review the most common failure cases of recent full hexahedral remeshing algorithms and show what we produce in these situations.

The most famous failure case is the jump ramp (Fig. 17) that make global parameterization fail with a field that has no singularity. It was observed (and partially fixed) in [Gregson et al. 2011] where the global parameterization was a simple polycube map. In this example, our method adds non hexahedral elements to smoothly connect 0 to $5 - 10$ layers of hexes.

Failure cases can also come from constraints that are not local (as in the lower part of the jump ramp). Such non-local constraints are typically encountered in the square screw example (Fig. 18). Considering a frame field topology without any singularity, finding a global parameterization is equivalent to constructing a polycube map. If we try to align all the surface normals with their closest axis, it will necessarily squish the model. In this configuration, our algorithm relaxes the incompatible constraints by introducing non hexahedral elements.

Beside these global issues, the input frame field is not always compatible with CubeCover, that requires a nice behavior of the frame field around its singularity curves. Local editing [Li et al. 2012] allows to filter-out noisy topology, but it is not always sufficient. For example, in Fig. 19 a singularity curve is doing half a turn inside the volume, that requires 3 different and incompatible orientation flags for the same curve. Our algorithm still works

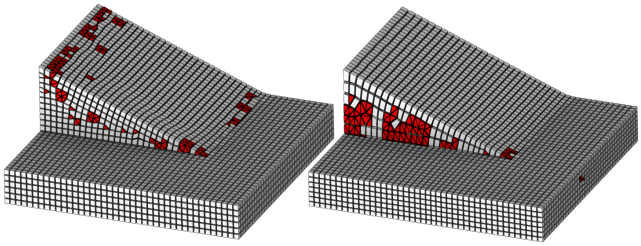


Fig. 17: Jump Ramp (a classical failure case of global parameterization method), treated by our method without (left) or with (right) scale correction. With scale correction, artifacts similar to the ones obtain with global parameterization methods are encountered.

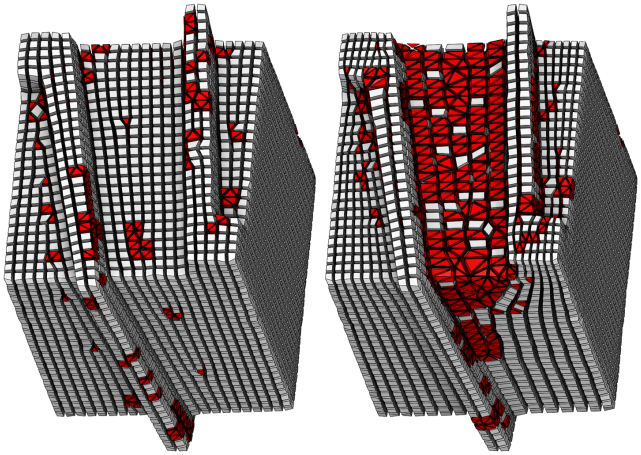


Fig. 18: Our algorithm without scale correction produces a nice hexahedral-dominant mesh for the squared screw (left). If we try to use scale correction, we have the same over constrained problem as CubeCover and it does not improve the result.

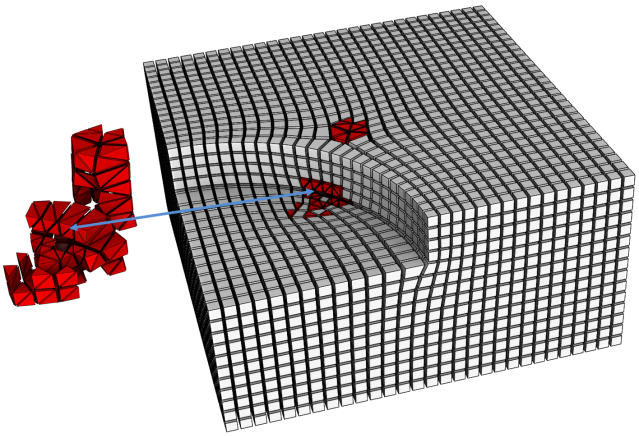


Fig. 19: Even if the frame field does not correspond to a full hexahedral mesh, our algorithm is able to produce a hexahedral-dominant mesh. The produced non-hexahedral elements are shown on the left.

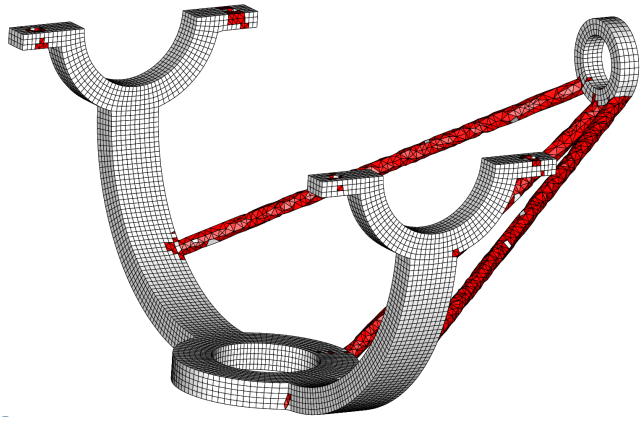


Fig. 20: Our algorithm is not able to produce hexahedra for small parts of the object (red), where it generates tetrahedra.

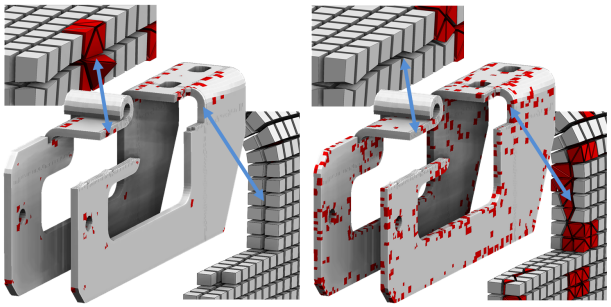


Fig. 21: The number of layers of hexahedra depends on the prescribed edge length. For a plate, this number of layers is constant (left). In some limit cases (right), the algorithm is not able to find the same number of layers everywhere on the model and "dithers" between two integer values, thus producing many singularities.

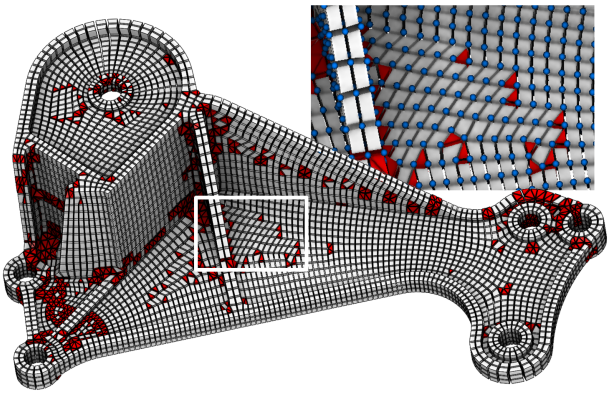


Fig. 22: When the pointset has too much shear, our recombination algorithm may find a (locally) better grid as shown in the close-up.

properly in these situations.

Our solution has drawbacks:

—When either the input mesh or the output mesh is not dense enough to represent the frame field singularity graph, the algo-

rithm is not able to produce hexahedra (Fig. 20). This drawback also exists in other methods;

—for plates and thick surfaces, our algorithm produces a small number of layers of hexahedra according to the desired edge length. For some particular values of the prescribed edge length, the number of layers is undetermined, in the sense that the optimization step produces a number of layers that varies on the model. Intuitively, the algorithm "dithers" between two number of layers, and these variations induce many singularities (Fig. 21). We mention that this phenomenon is seldom encountered, and producing such failure cases is difficult: changing the prescribed edge length by more than 1% solves the problem. However, by changing the prescribed edge value by up to 10%, one may still observe this behavior over limited local zones of the model, due to the input mesh resolution, as in Fig. 21—Left where a small number of tetrahedra is still produced (in red);

—whenever the orientation followed by the pointset is too much sheared, our recombination algorithm may produce hexahedra with a better geometry, but that are less coherent with their neighbors, as in the highligted zone of Fig. 22.

The possibility of creating non-hexahedral elements allows our algorithm to escape from the failure cases encountered with pure hexahedral meshing methods. In all the examples that we tested, our method generated a valid hexahedral dominant mesh in a fully automatic manner. The main drawback is that, in the cases listed above, our algorithm introduces non-hexahedral elements to avoid distortion of hexahedra, even in some cases where the distortion of a fully regular grid of hexahedra would remain acceptable.

Acknowledgments

The authors wish to thank one of the anonymous reviewers for pointing at a forbidden configuration in the recombination algorithm (configuration (3), bottom of page 10).

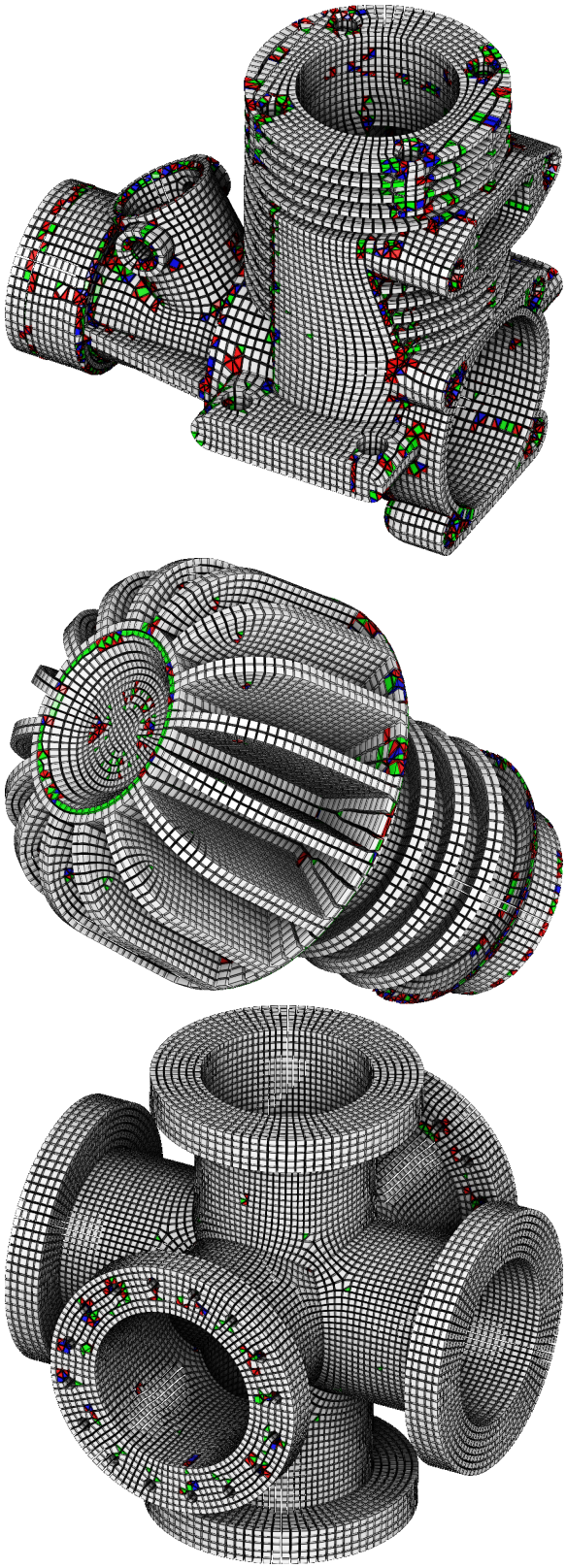


Fig. 23: Some results obtained from a database of models (continued next page, see also statistics in Table II and the supplemental material).

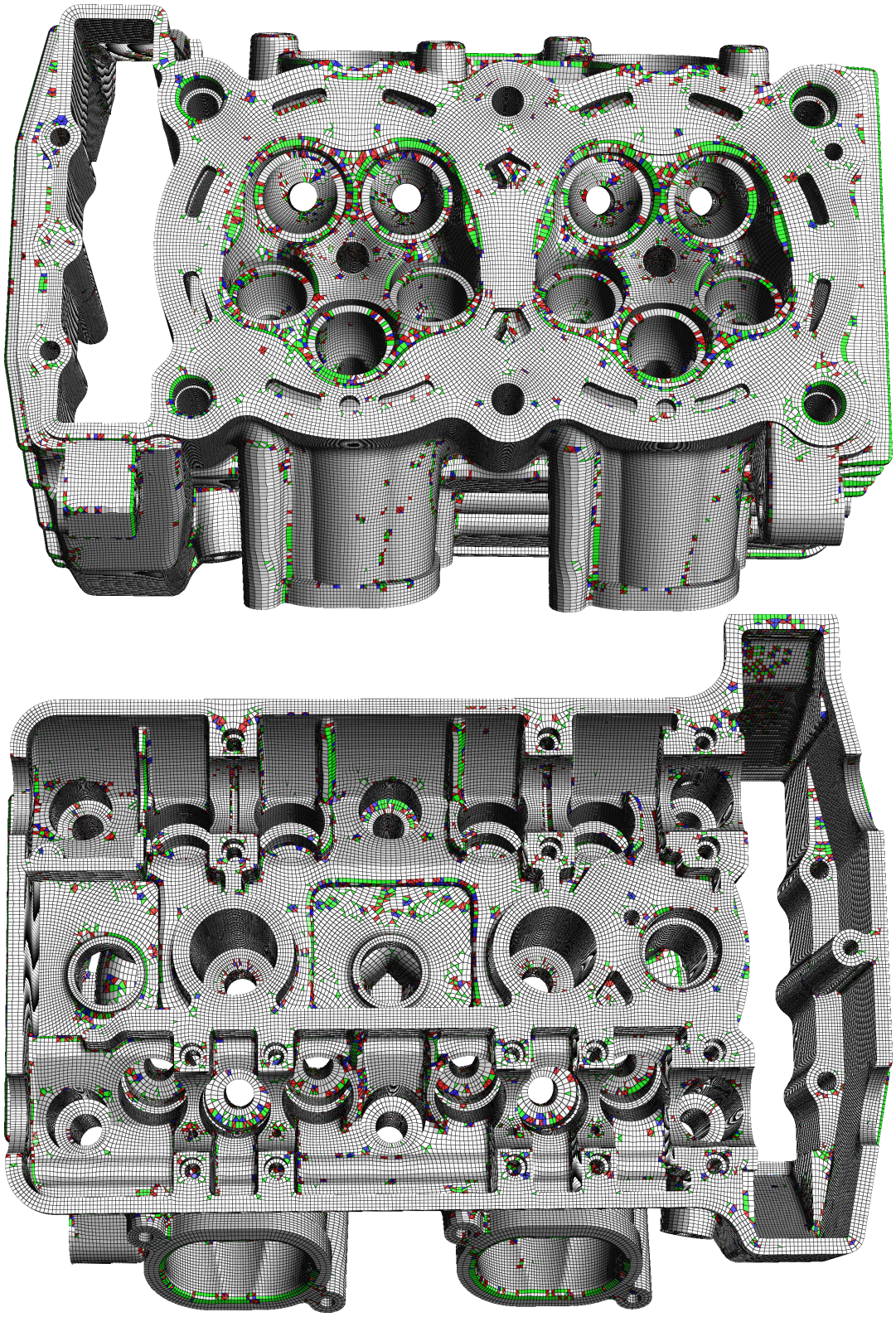


Fig. 24: Hexahedral-dominant mesh of the TRX engine (from GRABCAD). The input tetrahedral mesh has 2,123,979 vertices and 10,192,895 tetrahedra. The hexahedral-dominant mesh was generated by our algorithm in 1170 seconds (on a laptop with an Intel core i7 and 16Gb RAM). The result has 1,337,083 vertices and 1,894,549 cells in which 1,006,899 are hexahedra.

Table II. : Statistics and timings. #vert is the number of vertices of the generated model. The hexahedra proportion is measured by the percentage of hexahedra in number (H_{nbr}) and volume (H_{vol}). The hexahedra quality Q is measured by the scaled Jacobian in the format of average | standard deviation.

model	#vert	$H_{nb}\%$	$H_{vol}\%$	Q	input #tet	tot. time
piston	37899	89.87	97.28	0.98 | 0.03	771242	421.86
couplingdown	34543	66.63	93.69	0.98 | 0.05	1421348	831.41
cross 1	46221	89.07	97.63	0.99 | 0.03	451368	379.79
mazewheel	40762	67.55	89.28	0.97 | 0.05	652078	571.52
daratech	27639	61.97	87.57	0.96 | 0.05	785564	708.85
lego brick	40620	86.46	96.82	0.99 | 0.03	209991	174.42
front upright Assembly006	33787	89.75	96.56	0.98 | 0.03	1067311	649.63
Assembly Differential027	32324	35.27	74.12	0.91 | 0.08	1134100	793.88
david	32021	35.38	76.03	0.93 | 0.10	760922	651.86
socket	34457	91.54	97.89	0.98 | 0.03	1394945	706.03
champagne corck	38761	83.88	95.99	0.98 | 0.04	651298	549.6
gargo	29936	25.81	68.91	0.91 | 0.11	862287	707.52
Screw pump Inlet asm002	32714	82.56	94.05	0.98 | 0.04	631696	539.45
assy 4	32999	72.59	91.53	0.98 | 0.05	964657	694.05
pump	40625	89.94	97.56	0.98 | 0.03	714127	592.86
XR400 REAR HUB	40562	51.53	84.08	0.96 | 0.07	749564	672.04
block	39809	57.60	87.55	0.96 | 0.06	592920	685.02
ph4s3-mt	33536	80.18	96.55	0.98 | 0.03	1741821	822.17
Upright 6 3019	28031	50.15	84.57	0.94 | 0.07	1761305	1143.43
Cursore 02	35770	85.48	96.76	0.99 | 0.04	1214110	616.33
mohne	37575	68.15	93.59	0.98 | 0.06	2012917	1036.62
skull	33503	38.03	82.93	0.93 | 0.08	3791276	2429.86
gehaeuse	35685	87.11	97.17	0.99 | 0.03	1157572	618.28
Upright 6 3035	33850	82.63	94.24	0.97 | 0.04	795223	532.16
Screw pump asm003	38751	69.29	92.31	0.98 | 0.04	953105	638.69
Screw pump asm	44456	62.42	88.17	0.97 | 0.06	611669	701.39
gear2	42554	59.37	89.52	0.97 | 0.06	904855	566.99
Upright 6 3	38192	80.79	94.34	0.97 | 0.05	854942	586.99
fusee	36650	71.86	92.04	0.97 | 0.05	911345	640.35
venus	37258	29.59	79.11	0.93 | 0.09	4955207	5404.35
pump carter sup	34329	82.08	96.04	0.98 | 0.04	1210149	641.3
front upright Assembly008	42343	97.75	99.32	0.99 | 0.02	674573	427.2
boeing part	44196	85.79	95.82	0.98 | 0.04	556426	569.01
nasty cheese	44733	60.43	85.51	0.96 | 0.05	723396	835.43
monster	35394	73.81	94.12	0.98 | 0.05	1191511	652.03
front upright Assembly004	35154	81.89	96.01	0.98 | 0.04	1018336	523.5
front upright Assembly003	40169	77.21	94.86	0.98 | 0.04	774232	657.92
Assembly Differential003	40369	68.47	93.17	0.98 | 0.05	760916	566.19
Upright 6 3003	44938	82.06	95.57	0.98 | 0.04	759054	648.2
cochon	24034	47.67	83.60	0.93 | 0.07	224874	205.63
blower	45244	87.25	95.56	0.99 | 0.03	440560	291.15
cranckcse	41749	56.37	87.19	0.96 | 0.06	881380	979.69
front upright Assembly005	38354	63.72	88.75	0.97 | 0.06	775366	680.43
gear1	33110	93.08	98.29	0.99 | 0.03	1434621	769.94
bevel gear	29961	46.97	81.27	0.95 | 0.07	1518853	963.59
front upright Assembly013	29952	73.62	93.44	0.98 | 0.06	1757435	845.35
Assembly Differential026	36319	62.27	89.11	0.97 | 0.05	1000708	608.1
propeler jahte	34732	61.29	91.28	0.95 | 0.06	554109	626.84
Assembly Differential	30766	50.85	85.83	0.96 | 0.07	1588528	1039.63

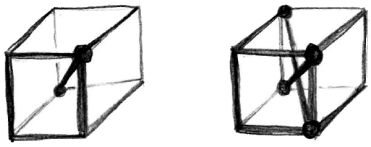
APPENDIX

A. ENUMERATING THE DECOMPOSITIONS OF A HEXAHEDRON

The main result in [Meshkat and Talmor 2000] (theorem 11 below) enumerates all the possible decompositions of a hexahedron into 5 or 6 tetrahedra. We explain here the main arguments used in the proof, identify some configurations that were not analyzed in the initial article and provide the additional arguments (Lemma 10) to rule them out. Our conclusions are the same as in the initial article, but comes with a complete proof (Appendix A.1). Our additional analysis provides the basis for studying the configurations with slivers (Appendix A.2). In [Botella et al. 2015], we previously gave an intuition that all configurations with slivers can be obtained by 'slicing' a hexahedron into two prisms, with the sliver 'sandwiched' between both prisms (besides the configuration obtained by 'gluing' slivers on faces of the hedahedron). We complement this intuition with a formal proof that no other configuration can exist. Before studying the configurations, we first give a lemma that bounds the total number of tetrahedra in a decomposition.

LEMMA 5. *The decomposition of a hexahedron into tetrahedra without any sliver glued on a quadrilateral face has at least 5 tetrahedra and at most 7 tetrahedra.*

PROOF. The decomposition satisfies the Euler-Poincaré identity, i.e. $\chi = V - E + F - T = 1$, where $V = 8$ denotes the number of vertices of the cube, E the number of edges in the decomposition, F the number of faces and T the number of tetrahedra. There are 12 facets on the border (two per quadrilateral facet), and 18 edges on the border (12 + 6 diagonals). Let F_{int} and E_{int} denote the number of internal facets and the number of internal edges respectively. We have $F = 12 + F_{int}$ and $E = 18 + E_{int}$. Note also that $4T = 12 + 2F_{int}$, or $F_{int} = 2T - 6$. Injecting these identities into the Euler-Poincaré identity, we get $T = E_{int} + 5$.



An internal edge is a diagonal of the cube, as shown on the left figure. In a cube decomposition, there can be at most two internal edges, configured like on the right figure (other configurations generate intersecting tetrahedra), therefore we have $0 \leq E_{int} \leq 2$, and $5 \leq T \leq 7$. □

Note that we supposed that there was no sliver glued on a face of the hexahedron. If we include configurations with such slivers, we can find up to $7 + 6 = 13$ tetrahedra in a decomposition(more on this in Appendix A.2).

A.1 Decomposition of a hexahedron into 5 or 6 tetrahedra

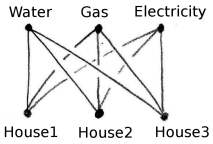
We now enumerate all the graphs that correspond to the decomposition of a hexahedron into five or six tetrahedra (and later, decompositions with up to 13 tetrahedra). To analyze the different possible decompositions, we use a graph representation, where each node corresponds to a tetrahedron, each solid arc corresponds to a facet shared by two tetrahedra, and each dashed arc corresponds to a pair of facets on the border that form a quadrilateral facet (see illustrations in §3.2.3).

The following lemmas are useful, since they exhibit some constraints that significantly restrict the set of graphs to be analyzed.

LEMMA 6. *The graph of the decomposition into five or six tetrahedra is planar.*

PROOF. A non-planar graph contains either the complete graph or the utility graph as a subgraph or a minor [Liu 1968] (i.e. can be transformed into the complete graph or the utility graph by deleting edges and nodes).

COMPLETE GRAPH



UTILITY GRAPH

Recalling that $F_{int} = 2T - 6$ (see proof of Lemma 5), the decomposition into 5 tetrahedra has $e = F_{int} = 4$ arcs and the decomposition into 6 tetrahedra has 5 arcs, which is smaller than the number of arcs in the complete graph and in the utility graph, therefore they do not contain any of them as a subgraph or a minor. □

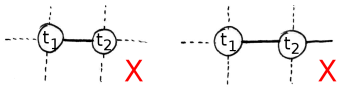
LEMMA 7. *The graph of the decomposition into 5 tetrahedra is a tree.*

PROOF. The 5 tetrahedra have 20 faces, including the 12 facets on the border of the hexahedra. The remaining 8 faces are internal. Since each solid arc corresponds to a pair of internal facets, there are 4 solid arcs in the graph. Since the graph is planar, the decomposition of the plane that it yields satisfies the Euler-Poincaré identity, i.e. $\chi = v - e + f = 2$, where v denotes the number of vertices in the graph ($v = T = 5$), e the number of arcs ($e = 4$) and f the number of faces (including the infinite face). In this case, we obtain $f = 1$, which means there is only the infinite face, thus the graph is a tree. □

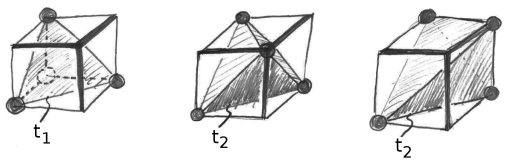
LEMMA 8. *The graph of the decomposition into six tetrahedra has exactly one cycle.*

PROOF. A derivation similar to the proof of Lemma 7 leads to $f = 2$, therefore the decomposition has the infinite face and another one, that corresponds to a cycle in the graph. □

LEMMA 9. *In the decomposition of a hexahedron into tetrahedra, if a tetrahedron has 3 facets on the border, its neighboring tetrahedron has either 0 or 1 facet on the border.*
In other words, the following configurations cannot appear in the graph:



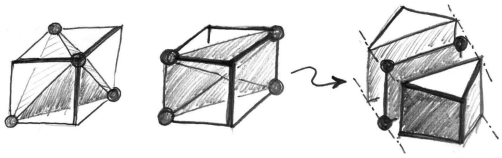
(where each dangling dashed edge corresponds to a tetrahedron facet on the border of the hexahedron).



PROOF. Let t_1 denote a tetrahedron with three facets on the border of the hexahedron (left figure). Let t_2 be a tetrahedron adjacent to t_1. There are only two possible configurations for t_2, with either no facet on the border (center figure) or one facet on the border (right figure). □

LEMMA 10. *In the decomposition of a hexahedron into 5 or 6 tetrahedra, if a tetrahedron has no facet on the border, then each of its 4 neighbors has 3 facets on the border.*
In other words, within the decompositions into 5 or 6 tetrahedra, only the following configuration has a tetrahedron with no facet on the border:

PROOF. By enumerating all the possible ways of choosing the 4 vertices of a tetrahedron from the 8 vertices of a hexahedron, one can see that the only two configurations where a tetrahedron does not have 3 vertices on the same facet of the hexahedron are as follows (left and center image):

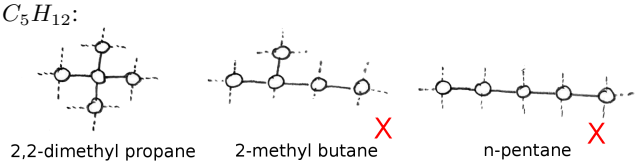


The configuration shown in the center image corresponds to a sliver that splits the hexahedron into two prisms (right image). Since the decomposition of each prism has at least 3 tetrahedra, this means that such a configuration can only appear in decompositions with at least 7 tetrahedra (more on this later). With 5 or 6 tetrahedra, only the configuration on the left can appear. Since there are exactly 12 triangular facets on the border of the hexahedron, each of the neighboring 4 tetrahedra has its remaining 3 facets on the border. □

Equipped with these lemmas, it is now simple to enumerate all the configurations with 5 and 6 tetrahedra. We find it worth mentioning that since all nodes are of degree 4, there is a natural correspondence between the set of admissible graphs and hydrocarbons. By looking-up all the isomers of C_6H_{12}, we found configurations that were overlooked in the initial article (in the end, the conclusion is the same, but this fills a hole in the proof). Each tetrahedron corresponds to a carbon atom, and facets on the border to hydrogen atoms. The configurations with 5 tetrahedra correspond to isomers of C_5H_{12}, and the configurations with 6 hexahedra to isomers of C_6H_{12} with one cycle:

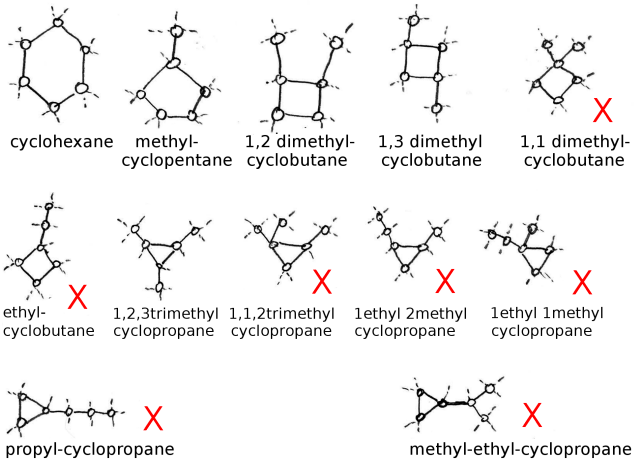
THEOREM 11. *There is exactly one possible decomposition of a hexahedron into five tetrahedra and there are exactly five decompositions of a hexahedron into six tetrahedra.*

PROOF. *Five tetrahedra:* There are three saturated isomers of C_5H_{12}:



Since they violate the condition in Lemma 9, the second one (2-methylbutane) and third one (n-pentane) do not correspond to the decomposition of a hexahedron.

Six tetrahedra: There are twelve isomers of C_6H_{12} with one cycle:

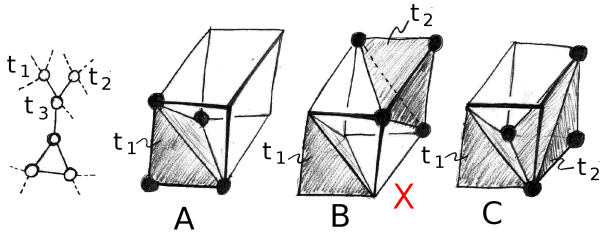


Among these twelve isomers, only five of them correspond to the decomposition of a hexahedron:

—ethyl-cyclobutane, propyl-cyclopropane, 1-ethyl-2-methyl-cyclopropane and 1-ethyl-1-methyl-cyclopropane are ruled-out by Lemma 9;

—1,1-dimethyl-cyclobutane and 1,1,2-trimethyl-cyclopropane are ruled-out by Lemma 10;

The last one (methylethyl-cyclopropane) requires a particularized analysis:



Tetrahedron t_1 has three facets on the border, and corresponds to a "chopped corner" of the hexahedron (A). For tetrahedron t_2, that also has three facets on the border, there are two possibilities (B,C). Configuration (B) is ruled out because there is no tetrahedron t_3 adjacent to both t_1 and t_2, therefore they must be in configuration (C). The only tetrahedron t_3 adjacent to both t_1 and t_2 has no facet on the border, which mismatches the "methylethyl-cyclopropane" graph on the left, where t_3 has exactly one facet on the border. Therefore there is no decomposition of a hexahedron into tetrahedra that matches this graph.

This completes the enumeration of all possible graphs and the discrimination of the ones that do not correspond to the decomposition of a hexahedron. □

A.2 Decomposition of a hexahedron into 7 to 13 tetrahedra

We now proceed to analyze the decomposition of a hexahedron into a larger number of tetrahedra, that include slivers, i.e. tetrahedra with nearly coplanar vertices. To introduce a sliver into the decomposition, there are two possibilities:

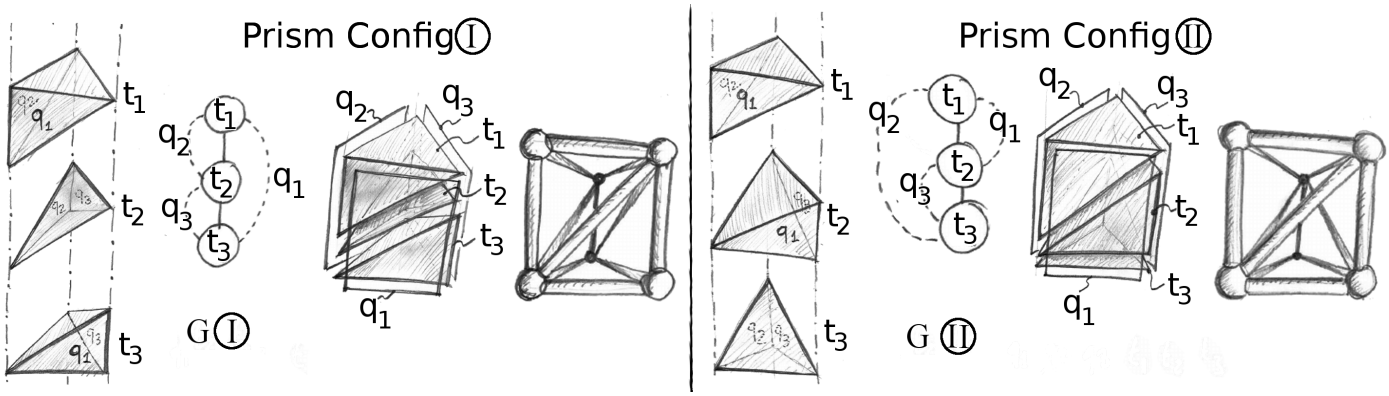
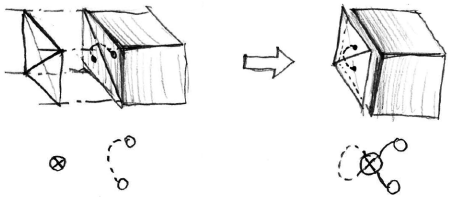


Fig. 25: For a fixed front quadrilateral face (q_1), there are two decompositions of a prism into three tetrahedra, from left to right, in exploded view, graph representation, compact view and shaded wireframe.

—append a sliver to a face of the hexahedron:



where the sliver is symbolized by a circled cross. The corresponding graph transform can be applied to each dashed arc, thus generating 2^6 graphs from each initial graph (modulo symmetries). We do not need enumerating all these graphs, it is easier to change the recognition algorithm as follows: once a hexahedron is recognized, we test whether a sliver can be merged to it for each of its faces;

—split the hexahedron into two prisms and connect them with a sliver (see the figure in the proof of Lemma 10).

This second way of introducing a sliver into the decomposition requires a finer analysis, since it more deeply changes the graph. Therefore, we need to enumerate all the new graphs that are generated by this operation. In other words, we need to identify within the five decompositions of a hexahedron into six tetrahedra which ones correspond to two prisms, and how the corresponding tetrahedra relate with the two prisms.

Without loss of generality, we consider the decomposition of a prism into three tetrahedra where the front quadrilateral facet q_1 and its diagonal are constrained, as shown in Fig. 25. In this setting, there are exactly two decompositions, G_I and G_{II}, that respect the constraint. Clearly the two graphs G_I and G_{II} are isomorphic (there is only one decomposition of a prism into three tetrahedra). What distinguishes G_I from G_{II} in this setting is which dashed arc corresponds to the constrained facet q_1.

At this point, one can notice that G_I is invariant by a 180 degrees rotation in the plane of the figure: the two vertices that are "far away" (black dots in the shaded wireframe view) are both of degree 4. In contrast, G_{II} is not rotation invariant: the two vertices that are "far away" are of degree 5 (top one) and 3 (bottom one). Therefore, G_{II} comes in two different "flavors", that we will call G^+_{II} if the degree 5 vertex is at the top (like on the figure), and G^-_{II} if the degree 5 vertex is at the bottom. Again, this does not make any difference when considering a single isolated prism, but it will generate different graphs when gluing two prisms along q_1.

We can now enumerate the different ways of creating a hexahedron by assembling two prisms in configuration G_I, G^+_{II} or G^-_{II}. As shown in Fig. 26, there are four different configurations, $G_I - G_I$, $G_I - G^+_{II} (= G_I - G^-_{II})$, $G^+_{II} - G^+_{II} (= G^-_{II} - G^-_{II})$ and $G^+_{II} - G^-_{II} (= G^-_{II} - G^+_{II})$. They correspond to four of the five decompositions of a hexahedron into six tetrahedra listed in the previous subsection. Inserting a sliver results in the graphs shown on the right column. The fifth configuration with six tetrahedra (1,2,3 trimethyl-cyclopropane) cannot be split into two prisms (and thus a sliver cannot be inserted into it).

This completes the enumeration of the decompositions of a hexahedron into 5 to 13 tetrahedra. To summarize, such a decomposition can be one of:

—One of the two "non-prismatic" decompositions into 5 and 6 tetrahedra (Fig. 27);

—one of the four "prismatic" decompositions (Fig. 26, left column);

—one of the four "prismatic" configurations with an internal sliver (Fig. 26, right column);

—a configurations obtained by appending 1 to 6 slivers to the quadrilateral faces of a configuration listed above.

REMARK 4. *With the same argument as in Lemmas 7 and 8, one can see that the graph of a decomposition into 7 tetrahedra has exactly two cycles, which is the case of the four configurations that we found (prismatic configuration with an internal sliver, right column of Fig. 26).*

B. GRAPH MATCHING ALGORITHM

This Appendix complements Section 3.2.3 with a complete description of the graph matching algorithm.

Once each of the 10 augmented graphs of Fig. 11 is encoded as a program, all the possible primitives in the input tetrahedral mesh T' can be recognized by the following algorithm:

Algorithm 3 finds each mapping $H = [t_0, t_1, \dots t_7]$ that maps local node indices in the template graph to global tetrahedra indices in the intermediate mesh T'. When such a mapping H is complete (then referred to as a *matched primitive*), it is appended to the set $\mathbf{H}^*$ of recognized primitives. A mapping H that is incomplete is referred to as a *matching state*. The program `Prg` is a list of operations, each operation being one of $\texttt{LINK}_{i,j}$ or $\texttt{QUAD}_{i,j}$. The two op-

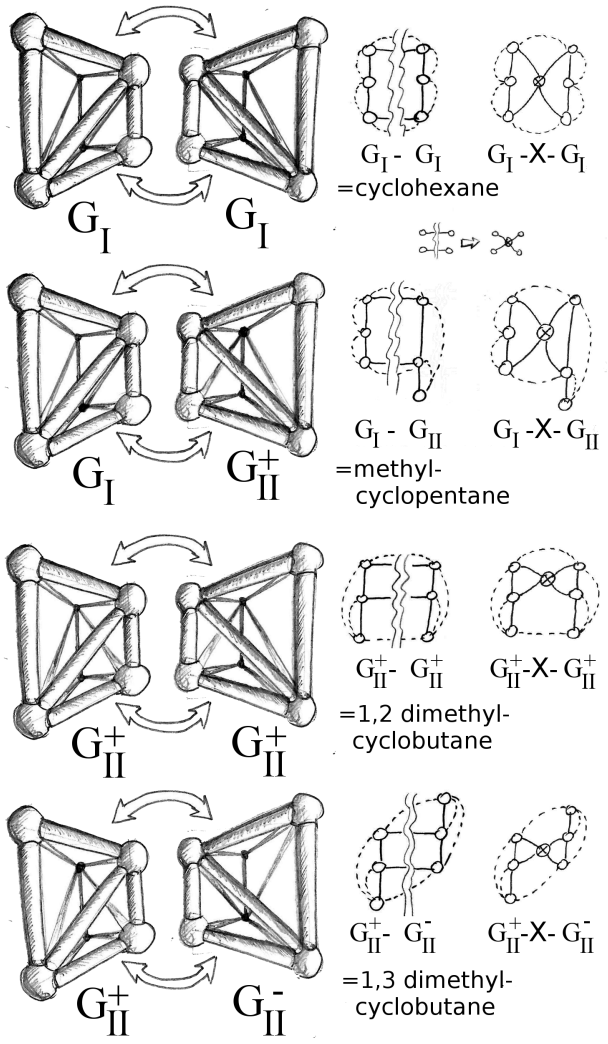


Fig. 26: There are four possible ways of assembling prisms of type G_I, G^+_{II} and G^-_{II}. For each configuration, a sliver can be inserted between the two prisms (graphs on the right).

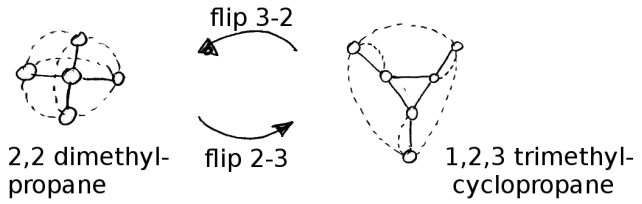


Fig. 27: Among the six decompositions into 5 or 6 tetrahedra, two of them cannot be split into 2 prisms. The one with six tetrahedra (right) can be deduced from the one with five tetrahedra (left) by applying a flip-2-3 operation to any pair of tetrahedra that share a facet.

1 FindAllMatches(Prg, T'):

Data: Prg: The program of an augmented graph; T': the intermediate tetrahedral mesh
Result: **H**: the set of all the primitives recognized by Prg in T'

2 **H** $\leftarrow \emptyset$
3 **foreach** $t \in T'$ **do**
4 | *// H is a "matching state", i.e. a local-node-index to*
5 | *// global-tet-index mapping, initialized with $t_0 = t$*
6 | H $\leftarrow [t, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset]$
7 | **H** $\leftarrow$ **H**$\cup$ FindMatchesRecursive(Prg, H, T')
8 **end**
9 FindMatchesRecursive(Prg, H, T'):

Data: Prg: Part of a program; H: a matching state; T': the intermediate tetrahedral mesh
Result: **H**: the set of all the instances recognized by Prg in T' from state H

10 **if** Prg $=$ EMPTYLIST **then**
11 | *// If* Prg *is empty, then all the instructions of the program*
12 | *// where consumed (H is a matched primitive)*
13 | MergeSlivers(H,T') ; **return** { H }
14 **else**
15 | *// Consume the first operation in the program*
16 | *//* Op *is one of* $\texttt{LINK}_{i,j}$ *or* $\texttt{QUAD}_{i,j}$ (see algo 4).
17 | Op $\leftarrow$ Head(Prg)
18 | PrgRest $\leftarrow$ Tail(Prg)
19 | *// The rest of the program is passed to* Op,
20 | *// to allow recursion / backtracking.*
21 | **return** Op(H, T', PrgRest)
22 **end**

Algorithm 3: The template-matching algorithm

erations are detailed in Algorithm 4 below. The function MergeSlivers detects all the slivers glued onto the 6 quadrilateral facets of the recognized hexahedron and merges them into the detected hexahedron. Such slivers are encountered whenever the pair of triangular facets that form a quad are connected to the same tetrahedron.

The $\texttt{QUAD}_{i,j}$ operation checks whether it is combinatorially possible to find a quadrilateral facet between tetrahedra t_i and t_j. The $\texttt{LINK}_{i,j}$ operation needs to explore all the possible assignments for tetrahedron t_j, resulting in a non-terminal recursion and requiring backtracking. In the algorithm, the facet f of a tetrahedron t_i is said to be *free* if tet_adjacent(T', t_i, f) is not in H ($\forall j \in [0..7], t_j \neq$ tet_adjacent(T', t_i, f)).

Implementation details / Optimizations: Since no inter-thread communication/synchronization is required, parallelization of the algorithm is very easy and directly gains a factor nearly linear in the number of cores. Besides this trivial optimization, we further optimized the algorithm, by early-discarding the matches that do not meet minimum quality requirements (for instance, a quadrilateral facet with angles that differ too much from 90 degrees). In addition, in the implementation of LINK, we avoid copying the matching state ($H' \leftarrow H$) when there is only a single facet of t_i that can be linked (i.e., when the recursion is terminal). Finally, to avoid unnecessary traversals of both T' and H, we keep track of the tetrahedron facets that are free by using a bitfield.

1 $\underline{\texttt{QUAD}_{i,j}(H,T',\texttt{PrgRest})}$:
Data: H: a matching state; T': the intermediate tetrahedral mesh, PrgRest: The rest of the program.
Result: $\mathbf{H}$: the set of recognized primitives
2 **if** t_i and t_j have two free facets with a common edge **then**
3 | **return** FindMatchesRecursive(PrgRest, H, T')
4 **else**
5 | **return** $\emptyset$
6 **end**

7 $\underline{\texttt{LINK}_{i,j}(H,T',\texttt{PrgRest})}$:
Data: H: a matching state; T': the intermediate tetrahedral mesh, PrgRest: The rest of the program.
Result: $\mathbf{H}$: the set of recognized primitives
8 $\mathbf{H} \leftarrow \emptyset$
9 **for** $f \in \{0,1,2,3\}$ **do**
10 | **if** CanLink$_{i,j,f}(H,T')$ **then**
11 | | $H' \leftarrow H$ *// Copy the state, for backtracking*
12 | | $t'_j \leftarrow$ adjacent_tet(T',t_i,f)
13 | | $\mathbf{H} \leftarrow \mathbf{H} \cup$ FindMatchesRecursive(PrgRest, H', T')
14 | **end**
15 **end**

16 $\underline{\text{CanLink}_{i,j,f}(H,T')}$
Data: i,j: two local tet indices; f: a facet index, H: a matching state; T': the intermediate tetrahedral mesh
Result: TRUE if i and j can be linked, FALSE otherwise
17 **if** $t_j = \emptyset$ **then**
18 | **return** facet f of t_i is free
19 **else**
20 | **return** adjacent_tet$(T',t_i,f) = t_j$
21 **end**

Algorithm 4: Algorithms for QUAD and LINK

REFERENCES

ASHBY, MANTEUFFEL, AND SAYLOR. 1990. A taxonomy for conjugate gradient methods. *J. Numer. Anal. 27*, 1542–1568.

BERGOT, M., COHEN, G., AND DURUFLÉ, M. 2009. Higher-order finite elements for hybrid meshes using new nodal pyramidal elements. *Journal of Scientific Computing 42,* 3, 345–381.

BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph. 28,* 3 (July), 77:1–77:10.

BOMZE, I. M., BUDINICH, M., PARDALOS, P. M., AND PELILLO, M. 1999. The maximum clique problem. In *Handbook of Combinatorial Optimization 4*. Kluwer Academic.

BOTELLA, A., LÉVY, B., AND CAUMON, G. 2015. Indirect unstructured hex-dominant mesh generation using tetrahedra recombination. *Computational Geosciences*, 1–15.

CARRIER-BAUDOUIN, T., REMACLE, J.-F., MARCHANDISE, E., HENROTTE, F., AND GEUZAINE, C. 2014. A frontal approach to hex-dominant mesh generation. *Advanced Modeling and Simulation in Engineering Sciences 1,* 1.

CGAL. Cgal open source project.

COOK, S. A. 1971. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symposium on Theory of Computing*. 151–158.

EBKE, H.-C., BOMMES, D., CAMPEN, M., AND KOBBELT, L. 2013. Qex: Robust quad mesh extraction. *ACM Trans. Graph. 32,* 6 (Nov.), 168:1–168:10.

EDELSBRUNNER, H. AND MÜCKE, E. P. 1990. Simulation of simplicity. *ACM TRANS. GRAPH 9,* 1.

GEORGE, P., HECHT, F., AND SALTEL, E. 1990. Fully automatic mesh generator for 3d domains of any shape. *IMPACT Comput. Sci. Eng. 2,* 3, 187–218.

GREGSON, J., SHEFFER, A., AND ZHANG, E. 2011. All-hex mesh generation via volumetric polycube deformation. *Computer Graphics Forum (Special Issue of Symposium on Geometry Processing 2011) 30,* 5, to appear.

HUANG, J., JIANG, T., SHI, Z., TONG, Y., BAO, H., AND DESBRUN, M. 2014. L1 based construction of polycube maps from complex shapes. *ACM Trans. Graph. 33,* 3 (June), 25:1–25:11.

HUANG, J., TONG, Y., WEI, H., AND BAO, H. 2011. Boundary aligned smooth 3d cross-frame field. *ACM Trans. Graph. 30,* 6 (Dec.), 143:1–143:8.

JIANG, T., HUANG, J., WANG, Y., TONG, Y., AND BAO, H. 2014. Frame field singularity correction for automatic hexahedralization. *IEEE Transactions on Visualization and Computer Graphics 20,* 8, 1189–1199.

KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2007. Quadcover – surface parameterization using branched coverings. *Computer Graphics Forum 26,* 3, 375–384.

LÉVY, B. AND LIU, Y. 2010. Lp Centroidal Voronoi Tesselation and its applications. *ACM Transactions on Graphics 29,* 4.

LI, Y., LIU, Y., XU, W., WANG, W., AND GUO, B. 2012. All-hex meshing using singularity-restricted field. *ACM Trans. Graph. 31,* 6 (Nov.), 177:1–177:11.

LIU, C. 1968. *Introduction to Combinatorial Mathematics*. Mc-Graw Hill.

LIVESU, M., VINING, N., SHEFFER, A., GREGSON, J., AND SCATENI, R. 2013. Polycut: Monotone graph-cuts for polycube base-complex construction. *Transactions on Graphics (Proc. SIGGRAPH ASIA 2013) 32,* 6.

LÉVY, B. 2000. OPENNL, Open Numerical Library. http://alice.loria.fr/index.php/software/4-library/23-opennl.html.

LÉVY, B. 2015. GEOGRAM, a programming library of geometric algorithms. http://alice.loria.fr/software/geogram/doc/html/index.html.

LÉVY, B. AND LIU, Y. 2010. Lp centroidal voronoi tesselation and its applications. *ACM Transactions on Graphics (SIGGRAPH conference proceedings)*. patent pending - FR 10/02920 (filed 07/09/10).

MESHKAT, S. AND TALMOR, D. 2000. Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh. *International Journal for Numerical Methods in Engineering 49,* 1-2, 17–30.

MEYER AND PION. 2008. FPG: A code generator. In *Real Numbers and Computers*. 47–60.

MEYERS, R. J., TAUTGES, T. J., TUCHINSKY, P. M., AND TUCHINSKY, D. P. M. 1998. The "hex-tet" hex-dominant meshing algorithm as implemented in cubit. In *in CUBIT; Proceedings, 7 th International Meshing Roundtable 98*. 151–158.

NIESER, M., REITEBUCH, U., AND POLTHIER, K. 2011. Cubecover– parameterization of 3d volumes. *Computer Graphics Forum 30,* 5, 1397–1406.

RAY, N., LI, W. C., LÉVY, B., SHEFFER, A., AND ALLIEZ, P. 2006. Periodic global parameterization. *ACM Trans. Graph. 25,* 4 (Oct.), 1460–1485.

RAY, N. AND SOKOLOV, D. 2015. On smooth 3d frame field design. *CoRR http://arxiv.org/abs/1507.03351*.

SHEPHERD, J. F. AND JOHNSON, C. R. 2008. Hexahedral mesh generation constraints. *Eng. with Comput. 24,* 3 (June), 195–213.

SHEWCHUK. 1997. Adaptive precision floating-point arithmetic. *Discrete & Computational Geometry 18,* 3.

SI, H. 2015. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw. 41,* 2 (Feb.), 11:1–11:36.

YAMAKAWA, S. AND SHIMADA, K. 2003. Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells. *International Journal for Numerical Methods in Engineering 57,* 15, 2099–2129.

YAN, D., LÉVY, B., LIU, Y., SUN, F., AND WANG, W. 2009. Isotropic remeshing with fast and exact computation of restricted voronoi diagram. In *ACM/EG Symposium on Geometry Processing / Computer Graphics Forum*.