Robust Quantization for Polycube Maps

François Protais, Maxence Reberol, Nicolas Ray, Etienne Corman, Franck Ledoux, Dmitry Sokolov

Robust Quantization for Polycube Maps

FRANÇOIS PROTAIS*, Université de Lorraine, CNRS, Inria, LORIA, France
MAXENCE REBEROL, Université catholique de Louvain, Belgium
NICOLAS RAY, Université de Lorraine, CNRS, Inria, LORIA, France
ETIENNE CORMAN, Université de Lorraine, CNRS, Inria, LORIA, France
FRANCK LEDOUX, CEA, DAM, DIF, France
DMITRY SOKOLOV, Université de Lorraine, CNRS, Inria, LORIA, France

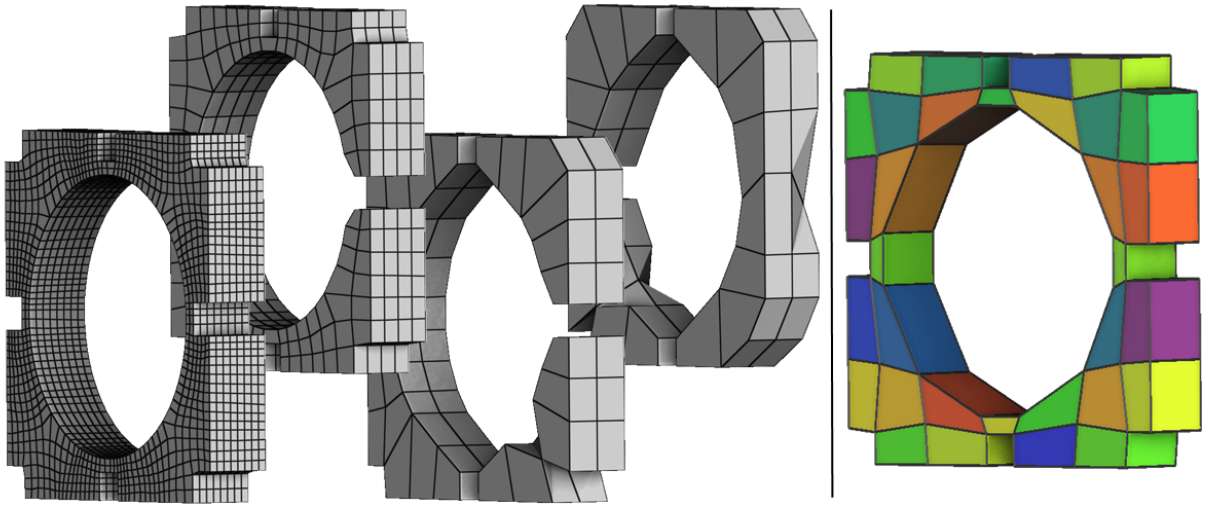

Fig. 1. Polycube-based hexahedral meshing algorithms are able to produce very regular grids (left), but fail to capture important geometric features with a coarse grid (left → middle). Our method allows to produce a coarse mesh preserving important geometric features (right).

An important part of recent advances in hexahedral meshing focuses on the deformation of a domain into a polycube; the polycube deformed by the inverse map fills the domain with a hexahedral mesh. These methods are appreciated because they generate highly regular meshes. In this paper we address a robustness issue that systematically occurs when a coarse mesh is desired: algorithms produce deformations that are not one-to-one, leading to collapse of large portions of the model when trying to apply the (undefined) inverse map. The origin of the problem is that the deformation requires to perform a mixed integer optimization, where the difficulty to enforce the integer constraints is directly tied to the expected coarseness. Our solution is to introduce sanity constraints preventing the loss of bijectivity due to the integer constraints.

CCS Concepts: • **Computing methodologies** → **Volumetric models**; **Mesh models**.

Additional Key Words and Phrases: hexahedral meshing, PolyCube-Map, rounding

1 INTRODUCTION

Hexahedral meshes exhibit superior performances than tetrahedral meshes in certain types of numerical simulations. They are commonly used for industrial applications like geological simulations, large deformations and non-linear elasto-plastic structural analyses [Dheeravongkit and Shimada 2006; Lessmann et al. 2018; Wheeler et al. 2012]. In practice, engineers make use of semi-automatic tools to precisely control cells orientation and quality. They often rely on coarse block decomposition of a domain which is then refined according to the simulation's need. A block decomposition allows them to generate a structured mesh, i.e. a mesh in which all internal nodes have identical connectivity, thus allowing to store little data and generate sparse FEM matrices [Armstrong et al. 2015]. Overall this process is very time consuming and can take days, weeks or even months.

Automatic hexahedral meshing is an active research area but remains a challenging task especially when looking for coarse block decompositions. One of the most promising approach comes from global parameterization. The idea is to design a 3D steering field prescribing the orientation of the hexahedra to produce, and then to compute a deformation of a regular grid with prescribed (local) orientation.

This approach faces two main obstacles: first, existing frame field-based methods cannot guarantee the validity of the singularity structure, resulting in degenerated parameterizations [Li et al. 2012; Nieser et al. 2011; Ray et al. 2016]. Second, current algorithms lack robustness when trying to produce meshes with arbitrarily large cells. In practice, mixed integer solvers often choose to allocate a

*Corresponding author: francois.protais@inria.fr

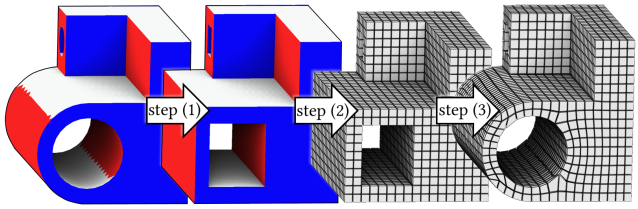

Fig. 2. Pipeline of hexahedral mesh generation with polycube.

negative or zero number of layers of hexahedra between two constrained sheets (boundaries of the object, internal constraints or singularities). Moreover, even having computed a parametrization, the extraction of the hexahedral mesh is a challenging problem [Lyon et al. 2016] as corrections of local non-bijectiveness of the map tend to break when generating coarse meshes.

One possible way to alleviate the first issue is to generate steering fields without (inner) singularities. This limitation trades some genericity against simpler and more robust algorithms. The idea is to deform the input shape into a polycube, i.e. a solid formed by joining multiple cubes of same size face to face [Gregson et al. 2011; Tarini et al. 2004].

In this article we focus on **robust** polycube generation enabling as coarse as possible polycube hexahedral remeshing. So, starting from a volume Ω, we want to compute a hexahedral mesh. Polycube methods are usually split into three parts (Figure 2):

(1) **Flagged deformation** — The boundary $\partial\Omega$ of the volume Ω is segmented into charts assigned with one of the six possible flags $\{\vec{u}, -\vec{u}, \vec{v}, -\vec{v}, \vec{w}, -\vec{w}\}$. A continuous, positive Jacobian mapping to the parametric space $\vec{u}, \vec{v}, \vec{w}$, is then computed such that its distortion is minimized and the image of each chart is planar and perpendicular to the axis of flagging. Formal definitions of the flagging and the map (to a *polycuboid*) are given in §3.1.

(2) **Quantization** — This second step makes sure that the faces of the polycuboid are aligned with the integer grid to obtain a *Polycube*. At this step, the volume can be filled by a unit grid.

(3) **Inversion** — The polycube deformation is inverted to extract the final hexahedral mesh.

All three steps have robustness issues, in this paper we address the quantization and the inversion problem. The first step (computing a flagging leading to a valid polycuboid) is very challenging [Sokolov and Ray 2015; Zhao et al. 2019], and no existing algorithm is guaranteed to produce a positive Jacobian polycuboid map. Solving these issues is an active research topic, that is orthogonal to the robustness issues treated in this paper. Therefore we assume that a valid polycuboid is given as an input.

So, given a continuous, positive Jacobian mapping $\mathfrak{g}$ of the object Ω to the parametric space, we do the following:

- We quantize the mapping, i.e. we find a valid set of integer boundary constraints.
- We compute the combinatoral structure of the hexahedral mesh induced by these integer constraints.

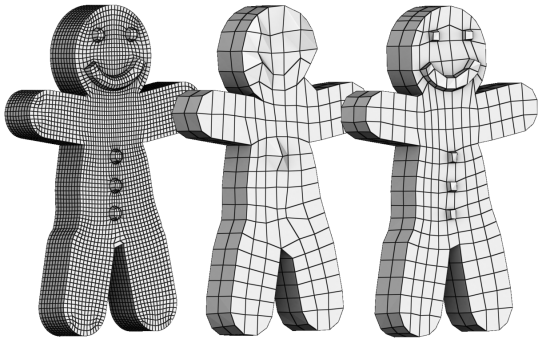

Fig. 3. Simply snapping variables to the nearest integer is not problematic for producing a high resolution mesh (Left), but becomes critical for a coarser mesh (Middle). Our constraints prevents such geometric collapses (Right).

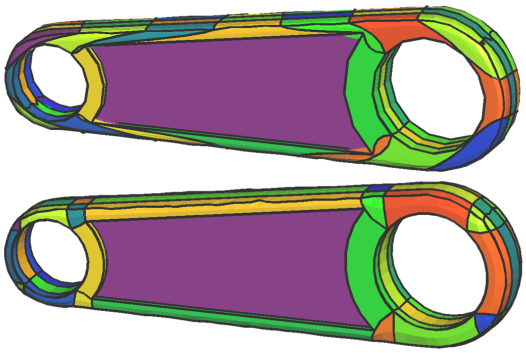

Fig. 4. A straightforward approach to compute the mapping of the interior is to first fix the boundary (initialization in [Livesu et al. 2013]). It produces highly distorted hexes when dealing with very coarse meshes (upper image). Moreover, hex mesh extraction may fail in presence of inverted elements. Our mappings have a much better geometry (bottom image) on the same mesh, and its extraction is more robust, because it does not rely on computing valid global parameterizations.

- We recover the geometry of the hexahedral mesh.

The usual approach to quantize the parameterization (step (2)) is to *snap* the polycube's faces to the closest integer coordinate plane. This is very damageable as several faces can be snapped together, leading to topological changes of the domain. This makes the inversion the deformation and extraction of a valid hexahedral mesh impossible. This technique typically collapses small geometric details of the model, but when looking for coarse polycubes it becomes so dramatic that the entire geometry is considered as a detail and collapsed to a point. *In this article, we provide theoretical and practical guarantees preventing collapsing (see Fig. 3).* Given a valid polycuboid (step (1)), we output a valid hexahedral mesh regardless of the level of coarseness.

To invert the polycube map (step (3)), the usual approach [Gregson et al. 2011; Livesu et al. 2013] is to compute the mapping as a piecewise linear map on a tetrahedral mesh of Ω. It was sufficient to

produce previous work results, but we are facing a more challenging situation: extremely coarse meshes imply much higher distortion (see Figure 4). To robustly handle this extreme situation, we compute the inverse map i.e. a transformation of the polycube to Ω. Namely, we extract the combinatorial structure of the resulting hexahedral mesh from the quantized map, and we optimize for its geometry. It removes the step of re-computing a global parametrization prone to have degeneracies/inverted elements that even HexEx [Lyon et al. 2016] would not be able to recover.

CONTRIBUTIONS

To sum up, we propose an approach which, given a positive Jacobian flagged deformation $\mathfrak{g}$ (step (1)), produces a valid hexahedral mesh, even for **very coarse meshes**. This paper contributes to three aspects of polycube-based hexahedral meshing:

- *We guarantee to quantize well the flagged deformation.* Namely, we construct a positive Jacobian parameterization with integer boundaries.
- *We generate hexahedral meshes in a robust way:* the positivity of the Jacobian for the input flagged deformation $\mathfrak{g}$ is not necessary, refer to §6.
- *We generate hexahedral meshes in a robust way:* our mesh generation does not rely on computing positive Jacobian global parameterizations for the inversion step.

2 PREVIOUS WORK

Polycubes were introduced in computer graphics for seamless texturing of triangulated surfaces [Tarini et al. 2004]. The first automatic algorithm for polycube generation [He et al. 2009] computes a scalar field on the object that constrains the z-coordinate of the deformed object. The remaining dimensions are solved by a 2D rasterization. Recent methods [Fu et al. 2016; Gregson et al. 2011; Huang et al. 2014; Zhao et al. 2018] prefer to optimize all the coordinates of the deformation simultaneously. The idea is to progressively deform the model in a way that each boundary triangle of the deformed object has one of its coordinates constant and integer. At each iteration, the triangle's normal becomes more and more aligned with one of the axes of the coordinate system. Each triangle would basically have to be aligned with the closest axis to its normal, but more elaborate strategies [Livesu et al. 2013] can improve the results. The final hexahedral mesh is typically improved by padding, possibly with a global optimization [Cherchi et al. 2019].

These algorithms have two major sources of failure. First of all, there may not exist a valid deformation aligning all triangles w.r.t a given flagging. Many failure cases can be fixed by heuristics [Fu et al. 2016; Gregson et al. 2011; Huang et al. 2014], but solving it in the general case is very difficult [Sokolov and Ray 2015] and beyond the scope of this paper. Second, respecting the flagging is only a half of the story: the integer coordinates of the boundary are to be chosen in a way that allows us to produce a valid hexahedral mesh. In this article we are focusing on the second issue.

The finer a hexmesh is, the easier it is to produce, so a natural way to approach the problem of coarse block decomposition is to generate a fine hexahedral mesh, and to decimate it with a general hexahedral mesh simplification algorithm [Gao et al. 2015, 2017]. At each step, the algorithm applies a simplification operation (sheets or chords collapse) to the hexahedral mesh. The geometry of the new mesh is obtained by a reparameterization.

It is also possible to obtain a coarse structure prior to the mesh generation phase, by working directly on coordinates of charts representing the polycube. Cherchi *et al.* [Cherchi et al. 2016] locally align a pair of polycube corners along one coordinate. Zhao *et al.* [Zhao et al. 2019] optimize the polycube edge lengths and Chen *et al.* [Chen et al. 2019] optimize the position of charts, but both consider only the boundary of the polycube, which is insufficient to prevent loss of bijectivity (as illustrated in Figure 13). Guo *et al.* [Guo et al. 2020] extend [Chen et al. 2019] approach, considering charts that are opposite to one another, but their constraints are still not exhaustive. Our algorithm has lower expectations on the input, and provides an optimal solution (instead of a greedy strategy), and is guaranteed to deliver a valid output.

The polycube approach can be extended by introducing cuts in the deformation to remove some unnecessary constraints, as Fang *et al.* [Fang et al. 2016] and Guo *et al.* [Guo et al. 2020] propose to do. Interestingly if we consider all possible cuts, we fall back to the family of hexahedral remeshing algorithms based on global parameterizations [Nieser et al. 2011]. Global parameterizations have robustness issues similar to those of polycube remeshing: frame fields problems replace flagging problems, and the requirement on the bijectivity of the deformation is replaced by the positivity of the Jacobian of the global parameterization. Frame field design is currently the bottleneck of the process and is an active research area [Li et al. 2012; Liu et al. 2018] and as in the polycube case, bijectivity becomes critical when a coarse mesh is desired.

It has to be noted that for the 2D case (i.e. quad mesh generation) [Kälberer et al. 2007; Ray et al. 2006], there exists robust algorithms based on global parameterizations [Campen et al. 2015; Myles et al. 2014]. The main ingredient of these algorithms is the decomposition of the domain into quad-shaped charts by cutting the domain along a motorcycle graph (for example, by tracing frame field streamlines). With this decomposition one can generate a quad mesh respecting the structure of these charts.

Our work can be understood as an extension of this approach to 3D. The fact that we restrict our attention to polycubes only simplifies the task: 3D motorcycle graphs are simply a set of axis-aligned planes in the parametric domain, the block decomposition has the combinatorial structure of a polycube (no T-junctions) and the edge lengths are manipulated by the coordinates of their extremities, which are directly the variables of the parameterization problem.

3 QUANTIZATION: PROBLEM STATEMENT

N.B. For the sake of clarity, we present the quantization problem in 2D; it is trivial to extend it to 3D (§6).

As mentionned earlier, parameterization methods give great results for quad mesh generation. The idea is to compute a parameterization f, or, in other words, two scalar fields f_1 and f_2 over the domain Ω (refer to Fig. 5). Integer iso-lines of these scalar fields form a structured quad mesh.

For a quad mesh to be well defined, these scalar fields must have following properties (refer to Fig. 6):

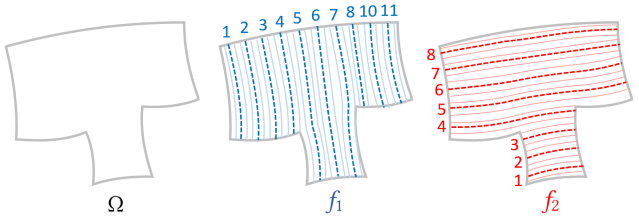

Fig. 5. Example of a parametrization $f = (f_1, f_2)$ of a domain Ω, displayed by its level-sets.

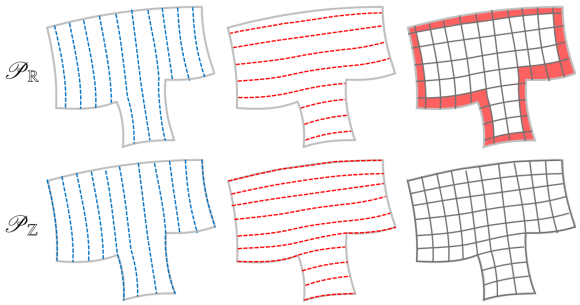

Fig. 6. **Top row:** a mesh extracted from a map in $\mathscr{P}_{\mathbb{R}}$ (refer to §3.1) has non boundary-aligned elements (in red). **Bottom row:** To obtain valid meshes, map in $\mathscr{P}_{\mathbb{Z}}$ must be considered.

(1) They must be well behaved: the Jacobian of the map f must be positive.
(2) They must be aligned with the boundary $\partial\Omega$, i.e. the boundary must be on an iso-line of one of the scalar fields.
(3) Moreover, on each point of the boundary $\partial\Omega$, at least one of the scalar fields must be *integer*.

In this work, we propose to start from a parametrization that respects the properties (1) and (2), but violates (3); we will compute another parameterization that holds all three properties. In §3.1 we introduce few notations necessary to formalize the problem (§3.2).

3.1 Notations

We denote the domain as $\Omega \subset \mathbb{R}^2$, and its boundary as $\partial\Omega$. Let $X = (x, y)$ be a point of $\mathbb{R}^2$. The Jacobian matrix of a parameterization $f : (x, y) \rightarrow (f_1, f_2) \in \mathbb{R}^2$ is denoted as $Df = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix}$, and its determinant as $\det(Df) = \frac{\partial f_1}{\partial x}\frac{\partial f_2}{\partial y} - \frac{\partial f_1}{\partial y}\frac{\partial f_2}{\partial x}$.

*Definition 1 (**Real polycube maps** $\mathscr{P}_{\mathbb{R}}$).* We say that the map $f : \Omega \rightarrow \mathbb{R}^2$ is in $\mathscr{P}_{\mathbb{R}}$ if:

- $f \in C^0\left(\Omega, \mathbb{R}^2\right)$ (continuity)
- $\det(Df) > 0$ (positive Jacobian)
- $\forall X \in \partial\Omega, f_1(X) \in L_1^r$ or (inclusive) $f_2(X) \in L_2^r$ (boundary is a set of iso-lines), where $L^r = \left\{L_1^r, L_2^r\right\}$ a finite set of **real** values.

*Definition 2 (**Integer polycube maps** $\mathscr{P}_{\mathbb{Z}}$).* We say that $f \in \mathscr{P}_{\mathbb{R}}$ is in $\mathscr{P}_{\mathbb{Z}}$ if $L^r \subset \mathbb{Z}$ (and we will denote $L^r = L$)

Let us take $f \in \mathscr{P}_{\mathbb{R}}$. The boundary $\partial\Omega$ can be broken down into a set of charts U_i for the first dimension and V_j for the second dimension. Charts are iso-lines of f such as $f_{1|_{U_i}} \in L_1^r$ and $f_{2|_{V_j}} \in L_2^r$.

To simplify the notations, we refer to these constant values as $f_{1|_{U_i}}$ and $f_{2|_{V_j}}$ (refer to Fig. 7). Throughout the paper, whenever possible, we put the expressions for the 1st dimension only, and we mention "resp V_j" to signify that it is trivial to retrieve the 2nd expression.

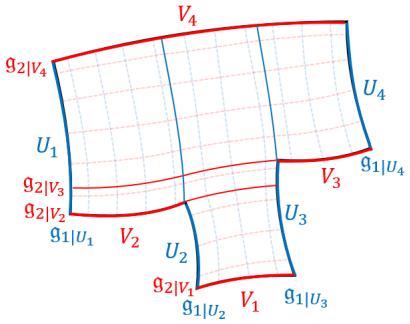

Fig. 7. Given a real polycube map $\mathfrak{g}$ over the domain Ω, the boundary $\partial\Omega$ is split into a set of charts U_i (resp. V_j); corresponding iso-values of $\mathfrak{g}_1$ and $\mathfrak{g}_2$ are denoted by $\mathfrak{g}_{1_{|U_i}}$ and $\mathfrak{g}_{2_{|V_j}}$.

3.2 Problem statement

Given an input real polycube map $\mathfrak{g} \in \mathscr{P}_{\mathbb{R}}$, the quantization problem is to find a similar (but integer!) polycube map $\bar{f} \in \mathscr{P}_{\mathbb{Z}}$, solution of:

$$\bar{f} = \underset{f \in \mathscr{P}_{\mathbb{Z}}}{\arg\min} \left\{ E(f) := \int_{\Omega} \|\nabla f_1 - \nabla \mathfrak{g}_1\| + \|\nabla f_2 - \nabla \mathfrak{g}_2\| dX \right\} \quad (1)$$

with $\|\cdot\|$ the Euclidean norm, and $E(f)$ is an energy measuring the distance between f and $\mathfrak{g}$.

A common approach to solve this problem is to greedily find a set L of new values for the boundary, such that $l_{U_i} \approx \mathfrak{g}_{1|_{U_i}}$ (resp. V_j). Then a new map constrained by these boundary values and minimizing (1) is computed. The problem, however, is that for some sets $L = \left\{ \left\{l_{U_i}\right\}_{i=1}^{|L_1^r|}, \left\{l_{V_j}\right\}_{j=1}^{|L_2^r|} \right\}$, it is impossible to define a smooth positive Jacobian polycube map i.e. the set $\mathscr{P}_{\mathbb{Z}}$ is empty. Thus the quantization fails. The trivial example is to set all constraints to zero ($l_{U_i} = 0, \forall i$); a quite common non-trivial failure case is shown in Fig 8.

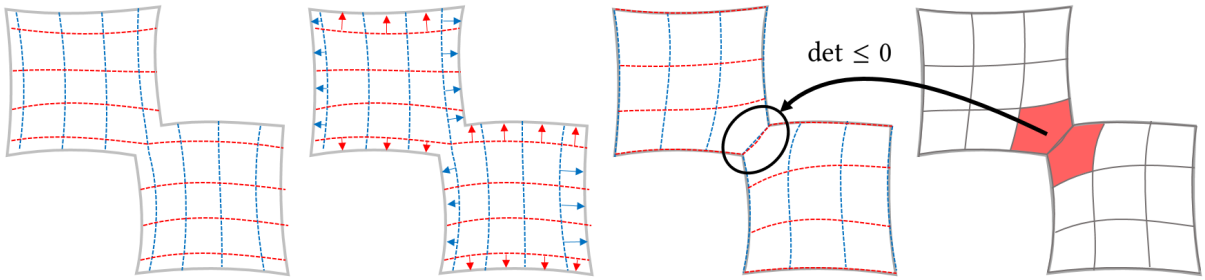

Fig. 8. Given a real polycube map (an element of $\mathscr{P}_{\mathbb{R}}$), a naive approach to compute an integer polycube map (an element of $\mathscr{P}_{\mathbb{Z}}$) is to round the boundary values to the nearest integer, and then to recompute the parameterization in order to spread out the resulting distortion. In certain cases it is impossible to generate a positive Jacobian integer polycube map, and thus the quad mesh extraction fails.

4 QUANTIZATION: OUR APPROACH

In this section we present a way to construct a map $F_L : \Omega \to \mathbb{R}^2$ for a given set of integer boundary constraints L. Then, in §4.1.1 and §4.1.2, we will introduce two constraints, [C1] and [C2], such that "L respects [C1] and [C2]" if and only if $F_L \in \mathscr{P}_{\mathbb{Z}}$. Armed with this construction, we approximate the problem (1) as follows:

$$\overline{f} = \underset{F_L:\, L \text{ respects [C1] and [C2]}}{\arg\min} E(F_L). \tag{2}$$

In the rest of the section we show the construction of F_L for a given set of integer boundary constraints L (§4.1) and we describe the optimization process in §4.2.

4.1 Construction of F_L

Recall that a triangulated domain Ω and a positive Jacobian **real** polycube map $\mathfrak{g} \in \mathscr{P}_{\mathbb{R}}$ are given as an input. First we show how to construct a function F_L for a given set $L \subset \mathbb{Z}$, and then we prove that F_L is a valid **integer** polycube map ($F_L \in \mathscr{P}_{\mathbb{Z}}$) if and only if L respects [C1]and [C2].

For each chart U_i (resp. V_j), we can extract the iso-lines of $\mathfrak{g}_1$ such that $\mathfrak{g}_1(X) = \mathfrak{g}_{1|_{U_i}}$. Since $\det(D\mathfrak{g}) > 0$, these iso-lines decompose the domain Ω into quadrangular blocks (see Fig. 9). To simplify a bit the input block decomposition, we ignore the connected components of the iso-lines that do not touch the corresponding chart (bottom row of Fig. 9).

Each block is mapped by $\mathfrak{g}$ to an axis-aligned rectangle limited by the values of four charts $U_i, U_{i'}, V_j, V_{j'}$ that delimited it. **N.B.** Without loss of generality, we consider that $\mathfrak{g}_{1|U_i} < \mathfrak{g}_{1|U_{i'}} \iff i < i'$ (resp. V_j).

Relying on this block decomposition, we define F_L as a sum of two functions:

$$F_L := h - \delta. \tag{3}$$

The underlying idea is to define the function h that is **almost** an integer polycube map. If L satisfies the constraint [C1], the function h is still a (valid) **real** polycube map ($h \in \mathscr{P}_{\mathbb{R}}$), but it maps the boundary $\partial\Omega$ very close to the integer values given by L. The function h does most of the work: it has a positive Jacobian ($\det(Dh) > 0$), and a small correction δ is needed to make $h - \delta$ a valid **integer** polycube map ($h - \delta \in \mathscr{P}_{\mathbb{Z}}$). This correction is only possible if L also satisfies the constraint [C2].

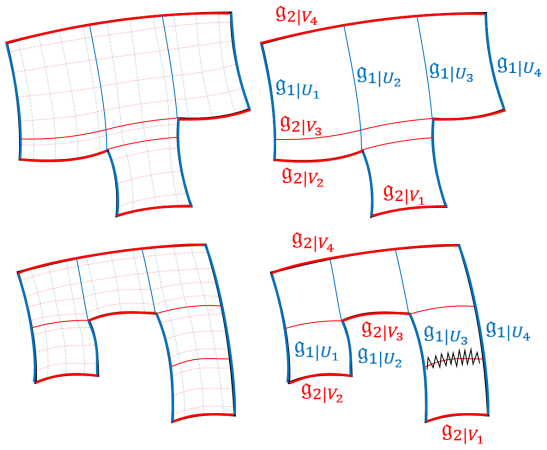

Fig. 9. The input real polycube map $\mathfrak{g}$ allows to decompose the domain Ω into quadrangular blocks. We do not extract connected components of the isolines that do not touch "corners" of the domain Ω (bottom row, right).

We represent h and δ by prescribing their values at the vertices of the block decomposition. Refer to Fig. 10: given a block with four vertices p^{11}, p^{12}, p^{21} and p^{22}, it is mapped to an axis-aligned rectangle by $\mathfrak{g}$. If we want to evaluate a function f represented by four values $f(p^{11}), f(p^{12}), f(p^{21})$ and $f(p^{22})$, then for any point p of the block we can find its barycentric coordinates $\lambda_1, \lambda_2, \mu_1, \mu_2$ w.r.t. the axis-aligned rectangle, and $f(p)$ can be evaluated as follows:

$$f(p) = \sum_{i,j \in \{1,2\}} \frac{\lambda_i \mu_j}{(\lambda_1 + \lambda_2)(\mu_1 + \mu_2)} f(p^{ij})$$

So, the block decomposition allows us to represent continuous blockwise bi-linear functions; h (§4.1.1) and δ are (§4.1.2) defined in this way.

4.1.1 Construction of h. As mentionned previously, we represent h by specifying its values at the vertices of the block decomposition.

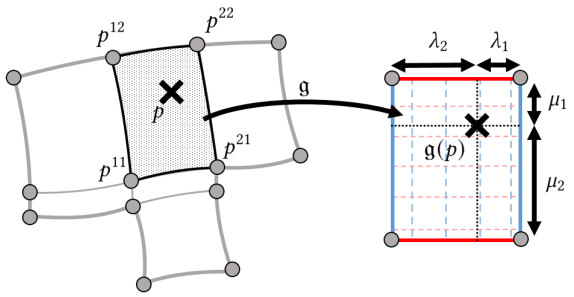

Fig. 10. Each quadrangular block is mapped to an axis-aligned rectangle by $\mathfrak{g}$. For every point p in a block we can find barycentric coordinates of its image $\mathfrak{g}(p)$ w.r.t. the axis-aligned rectangle; this allows us to define bi-linear functions over the original block by prescribing 4 values at the block vertices.

ALGORITHM 1: Computation of δ

Result: A value of $\delta(s)$ for each vertex s of the block decomposition
1 **for** $s \in B$ **do**
2 | $\delta_1(s) = 0$; `visited[`s`] = false`;
3 **for** $s \in U_i$ **do**
4 | $\delta_1(s) = i\varepsilon$; `visited[`$s$`] = true`;
5 **while** *There is an edge* (s,t) *such that* $\|h(s) - h(t)\| < |L|\varepsilon$ *and* `visited[`s`]` *and not* `visited[`t`]` **do**
6 | $\delta_1(t) = \delta_1(s)$;
7 | `visited[`t`] = true`;
8 δ_2 is computed the same manner;

Thus, for every vertex generated by two charts U_i and V_j, we define h to be equal to $(l_{U_i} + i\varepsilon,\ l_{V_j} + j\varepsilon)$ with $\varepsilon > 0$ and the bi-linear interpolation does the rest of the job.

It is easy to verify that $\det(Dh) > 0$ (refer to Appendix A) if L satisfies the constraint [C1] expressed as follows:

Definition 3. we say that L respects [C1]if :
Given a block limited by U_i and $U_{i'}$ (resp. V_j):

$$i < i' \Rightarrow l_{U_i} \leq l_{U_{i'}} \qquad ([C1])$$

Thus, under the constraint [C1]the mapping $h \in \mathscr{P}_{\mathbb{R}}$, i.e. it is a positive Jacobian real polycube map; it is easy to see that it is **almost integer**: it differs only by $O(\varepsilon)$ from the target, and only a small correction δ is needed so that $F_L := h - \delta$ is an **integer** polycube map.

4.1.2 Construction of δ. We need to compute a small correction function δ such that:

(1) $h - \delta$ has desired integer values L on the boundary $\partial\Omega$;
(2) the Jacobian $\det(DF_L)$ is positive.

Like h, the function δ is represented by its values on the vertices of the block decomposition, and is interpolated elsewhere. Alg. 1 shows how to compute the values by iterating over the edges of the block decomposition.

N.B. In our implementation we do not actually compute δ *everywhere*, but only for the boundary; nevertheless, we provide its explicit construction because it allows us to state the constraint [C2], which is useful to avoid degeneracies.

The construction is straightforward: first we guarantee that $h - \delta$ has the desired values L on the boundary (lines 3–4 of Alg. 1). Then, to ensure the positivity of the Jacobian $\det(D(h - \delta)) > 0$, we perform a front propagation (lines 5–7) over the edges of the block decomposition starting from each chart. The idea is to have all variations of δ situated where the variation of h is sufficient to keep the positivity of the Jacobian.

More precisely, our algorithm guarantees that if an edge (s,t) of the block decomposition is "short" under action of h (i.e. $\|h(s) - h(t)\| < |L|\varepsilon$), then the correction δ keeps its length unchanged, i.e. $\delta(s) = \delta(t)$. In Appendix B we demonstrate that this condition implies the positivity of the Jacobian $\det(D(h - \delta)) > 0$.

There is one caveat though. The algorithm propagates fronts starting from the charts, the propagation is made along "short" edges of the block decomposition. What happens if two different fronts, say, from charts U_i and U_j meet across an edge? By construction, that would imply that there is a path made entirely of "short" edges that links a vertex of U_i with a vertex of U_j. It implies the equality of integer boundary constraints $l_{U_i} = l_{U_j}$ and $l_{V_{i'}} = l_{V_{j'}}$; in turn, it implies that it is impossible to build a valid integer polycube map represented as a bi-linear interpolation over the block decomposition.

Fig. 8 provides a typical example of the violated constraint: two different points of Ω are placed in the same place in the parametric space, leading to a degenerate Jacobian. To avoid this, we put constraints on the set L: we force certain paths over the edges of the block decomposition to contain "long" edges. It is sufficient to consider only monotonic paths, i.e., paths that do not go back and forth in each of the dimensions.

Definition 4. We say that a path P over the edges of the block decomposition is *monotonic* if its coordinate functions $h_1(P)$ and $h_2(P)$ are (non-strictly) monotonic.

Thus, we put a constraint that every monotonic path between any two points of different charts in the same dimension, must contain a "long" edge. We call this constraint [C2] and it is formulated as follows:

Definition 5. We say that L respects [C2] if:
Given two vertices issued from charts U_i, V_j and $U_{i'}, V_{j'}$. If the vertices are both situated on same dimension charts and if there exists monotone path between them, then:

$$\text{sign}(i' - i)(l_{U_{i'}} - l_{U_i}) + \text{sign}(j' - j)(l_{V_{j'}} - l_{V_j}) > 0 \qquad ([C2])$$

Let us sum up: given a positive Jacobian real polycube map $\mathfrak{g} \in \mathscr{P}_{\mathbb{R}}$ and a set of integer constraints L; if L respects [C1] and [C2], then F_L, whose construction we give explicitly, is a valid positive Jacobian integer polycube map ($F_L \in \mathscr{P}_{\mathbb{Z}}$).

4.2 Estimating $E(F_L)$ and optimization

Having defined a construction of a function P_L for a given set of integer constraints L, we need to evaluate the energy $E(F_L)$ to be able to solve our optimization problem (2).

For a given value of ε, evaluation of $E(F_L)$ is quite cumbersome. Fortunately, we can notice the following fact:

$$\lim_{\varepsilon \to 0} E(F_L) = \lim_{\varepsilon \to 0} E(h)$$

The behavior of $E(h)$ at the limit with $\varepsilon \to 0$ is easy to study, some straightforward calculations (see Appendix C) give following expression:

$$\lim_{\varepsilon \to 0} E(F_L) = \sum_{b \text{ block of } \Omega} \left[\begin{array}{l} \alpha_1 \left| l_{U_{i'}} - l_{U_i} - (\mathfrak{g}_{1\,|U_{i'}} - \mathfrak{g}_{1\,|U_i}) \right| \\ +\alpha_2 \left| l_{V_{j'}} - l_{V_j} - (\mathfrak{g}_{2\,|V_{j'}} - \mathfrak{g}_{2\,|V_j}) \right| \end{array} \right], \quad (4)$$

where α_1 and α_2 are constant per block:

$$\alpha_1 = \frac{\int_b \|\nabla \mathfrak{g}_1(X)\|\, dX}{|\mathfrak{g}_{1\,|U_{i'}} - \mathfrak{g}_{1\,|U_i}|} \qquad \alpha_2 = \frac{\int_b \|\nabla \mathfrak{g}_2(X)\|\, dX}{|\mathfrak{g}_{2\,|V_{j'}} - \mathfrak{g}_{2\,|V_j}|}.$$

It allows us to reformulate the problem (2) as follows:

$$\overline{f} = \underset{F_L:\, L \text{ respects } [C1] \text{ and } [C2]}{\arg\min} \lim_{\varepsilon \to 0} E(F_L). \quad (5)$$

Under a proper change of variables, this optimization problem boils down to a mixed-integer linear program, i.e. a minimization of a linear energy under linear and integer constraints.

N.B. This problem always has a solution; as a matter of fact, the choice $l_{U_i} = i$ (resp. $l_{V_j} = j$) is a feasible set of constraints.

5 IMPLEMENTATION

For the sake of clarity, so far our method was presented in 2 dimensions, but it is straightforward to extend it to 3D. We discuss our implementation and results in 3 dimensions.

It is important to notice that the construction we proposed in Section 4 relies on a (local) bijectivity of $\mathfrak{g}$ everywhere over the domain. In practice, it is *a lot* to ask: usually real polycube maps have small defects that undermine our construction, particularly if we actually wanted to cut the input mesh along the iso-surfaces generated by the charts in order to extract the block decomposition.

To circumvent this issue, we **do not** compute the iso-surfaces, but rather we:

- § 5.1: extract the **combinatorial structure of the block decomposition** by rasterizing the image of Ω by $\mathfrak{g}$;
- § 5.2: this information suffices to deduce the **combinatorial structure of the resulting hex mesh** by solving the mixed-integer linear program given by Eq. (5);
- § 5.3: finally, we use F_L to recover the **geometry of the hex mesh**. Note that with the information in our hands, F_L^{-1} is not defined everywhere in the domain, however it is possible to evaluate it at the vertices of the hex mesh.

5.1 Extracting the block decomposition

As pointed out earlier, $\mathfrak{g}$ often presents small defects like inverted or degenerated tetrahedra. It is not an issue as long as at least all the charts are well defined and the image $\mathfrak{g}\left(\{U_i\} \bigcup \{V_j\} \bigcup \{W_k\}\right)$ forms a valid boundary of a volume. In this case, we can cut $\mathbb{R}^3$ with axis-aligned planes placed at the values $\mathfrak{g}_{1\,|U_i}$ (resp. V_j, W_k). This defines an axis-aligned grid in the parametric space, and by a simple rasterization we can find whether a current grid cell is in the image of Ω by $\mathfrak{g}$, or out. Having extracted all inner blocks, we obtain the combinatory structure of the block decomposition.

5.2 Computing hexahedral combinatory

The combinatorial structure of the resulting hexahedral mesh is completely defined by the quantization L, and it can be found by solving Eq. (5). We solve the mixed-integer linear program by using [CPLEX 2019], and the implementation is rather straightforward, but we must pay attention to two details:

- the proposed energy (Eq. (5)) relies on the computation of weights α_i, which requires to integrate $\nabla \mathfrak{g}_i$ over each block of Ω. Note that for robustness reasons we have chosen not to compute the block decomposition of Ω explicitly, therefore we approximate α_i in each block $b \in \Omega$ by the volume of the block under action of $\mathfrak{g}$, which is readily available.
- the constraint [C2] represents a lot of inequalities ($O(N^2)$, where N the number of vertices of the block decomposition). In our implementation, we treat them as "lazy" constraints. More precisely, we solve the problem without enforcing entirely [C2], *i.e.* we start by only constraining charts that are opposite to one another, similarly to [Guo et al. 2020]. Then, if the solution violates [C2], we add the "least respected" inequality to the problem, and restart until we find a solution that respects [C2]. In practice we never encountered a case where more than few iterations were necessary, refer to the right column of Table 1.

To compute the final combinatorial structure of the resulting hexahedral mesh, first we create the hexahedral mesh (still not endowed with geometry) inherited from the block decomposition. Then we remove or split hexahedra based on the quantization L.

So far we have computed the connectivity of the output hexahedral mesh but we are still missing the vertex positions. In §5.3 we will apply F_L^{-1} to find the final mesh.

5.3 Computing hexahedral geometry

The last step of the process is to compute the geometry of the final hexahedral mesh. We have already computed its image under F_L, and therefore we need to apply F_L^{-1} to the previous result.

The problem, however, is that we have chosen to compute only the combinatorial structure of the block decomposition and not its geometry in Ω; therefore F_L is not defined everywhere in the domain with the information we have in our hands. Fortunately, it is well-defined for the vertices of our hexahedral mesh. Applying F_L^{-1} to the vertices of the hexahedral mesh boils down to mapping the mesh onto $\mathfrak{g}(\Omega)$ and then to applying $\mathfrak{g}^{-1}$ to the vertices. Since in practice $\mathfrak{g}$ may present local imperfections (e.g. foldovers), the resulting mesh might not be perfectly conforming with the boundary, or possess poor quality elements. In this case, we can apply a mesh optimisation algorithm such as [Livesu et al. 2015] to improve the boundary compliance and get rid of poor quality elements.

6 RESULTS

We tested our algorithm on a i7 4.3GHz CPU with 64GB of RAM on a single thread implementation. We compute the initial real polycube map $\mathfrak{g}$ with the algorithm [Gregson et al. 2011].

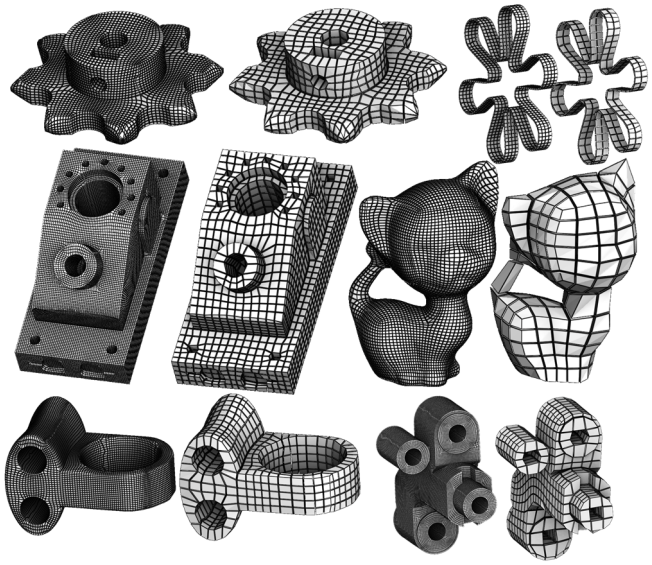

Fig. 11. Our robust quantization process allows for multi resolution meshes, without risk of collapses of the geometry.

Id	Model name	size (tets)	# blocks	# hexahedra	total time (sec)	CPLEX calls
1	kiss	302292	2558	410	150	2
2	Bunny	731085	426	58	130	2
3	grid	6718	285	56	2.1	1
4	street	2327	483	55	3.0	3
5	connecting rod	14380	203	54	2.6	1
6	engine1	18864	6151	219	4.6	1
7	engine2	56789	9415	404	9.2	1

Table 1. Statistics for the models shown in Fig 12.

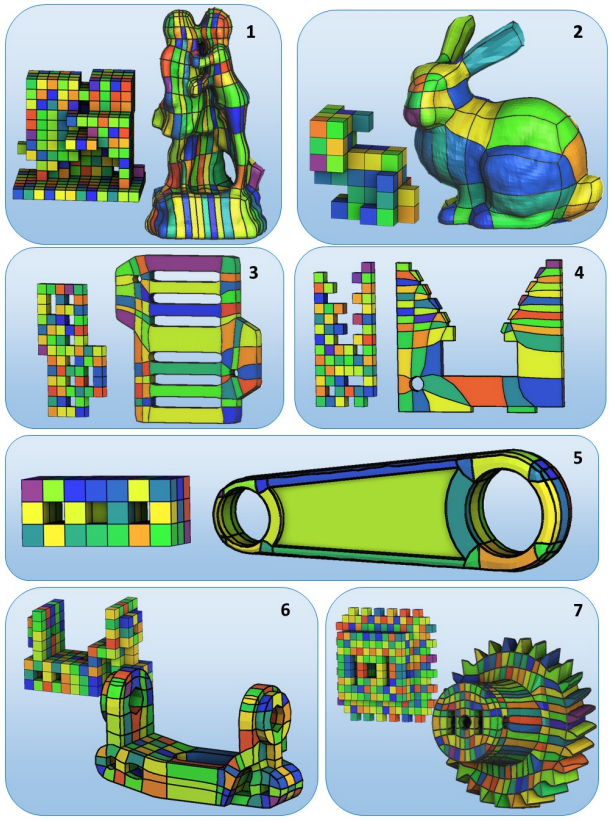

Fig. 12. For each model, we optimize the block decomposition to get an as coarse as possible polycube. The associated mesh will give a very coarse decomposition of the original domain Ω.

6.1 Hexahedral meshing

We use our algorithm to generate hexahedral meshes minimizing the distortion with respect to the initial real polycube map $\mathfrak{g}$. The resolution of the meshes is determined by the scale of the input real polycube map $\mathfrak{g}$. Fig. 11 shows the results for two different resolutions. Unlike the naive snapping technique sometimes used for quantization (Fig. 3), our quantization produces valid hexahedral meshes (no missing hexes) without loss of geometric features. We ran our code on 1000+ models from the dataset [Hu et al. 2018]. Results are provided in the supplementary material[1]. Concerning our claims on robustness to local non-bijectivity, it should be noted that **from the 1391 model tested, 815 (59%) had at least one flipped tetrahedron** through $\mathfrak{g}$.

6.2 As coarse as possible block decomposition.

To challenge the robustness of our algorithm, we generate as coarse as possible hexahedral meshes by using a down-scaled $\mathfrak{g}$, *i.e.* , replace the input $\mathfrak{g}$ by $\eta\mathfrak{g}$ with $\eta \approx 0$. The result can be interpreted as a block decomposition that minimizes the distance between pairs of charts. Table 1 provides statistics on 7 models, the decompositions are shown in Fig. 12. Note that $\mathfrak{g}$ was computed with [Gregson et al. 2011] for all models except the two first models from Table 1 that come from the supplemental material of [Livesu et al. 2013].

We reported the number of blocks in the input polycuboid, the number of blocks in the output generated with global overlaps as well as the timings and the number of times we had to call CPLEX. It can be seen that we achieve a significant simplification of the mesh structure, and that our algorithm is robust to extreme simplifications, being able to recover valid meshes even for highly distorted polycube maps.

Comparison with polycube simplification methods. [Zhao et al. 2019], [Chen et al. 2019] and [Guo et al. 2020] propose a set of constraints to guarantee the validity of the polycube. The problem, however is that their constraints are not sufficient and can potentially lead to degenerate polycube during simplification. They only guarantee the integrity of the boundary of the polycube, missing the deformation *inside* the domain. Fig 13 shows a failure example: the bottom of the cup collapses into a plane without violating their constraints, whereas our set of constraints guarantees the validity of the polycube map. Cherchi *et al.* [Cherchi et al. 2016] attempt to locally align corners of the polycube. As a consequence they fail to align vertices that are far away from each other. In contrast, our method considers all possible chart alignments. For example, the

[1]Contact: francois.protais@inria.fr

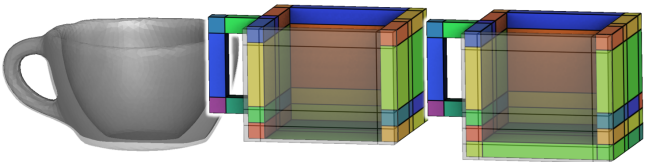

Fig. 13. Quantization without volumic constraints [Chen et al. 2019; Zhao et al. 2019] can not detect the collapse of the bottom of the cup (middle). We take them into account (right). [Guo et al. 2020] would succeed on this model, but would fail on more tricky set-up (such as Fig 8).

	# of blocks			Time (min)	
	Original	GAO et al. 2017	Us	GAO et al. 2017	Us
Top	33	23	28	27	<1
Middle	347	80	80	>200	<1
Bottom	70	24	16	24	<1

Table 2. Comparison with mesh simplification method [Gao et al. 2017]: number of blocks before and after simplification and timing. Models are rendered in Fig 14. Our method leads to similar number of blocks with a considerable speed-up (~ 20×). For the top model Gao *et al.* has fewer blocks only because they allow non-polycube deformation (close-up).

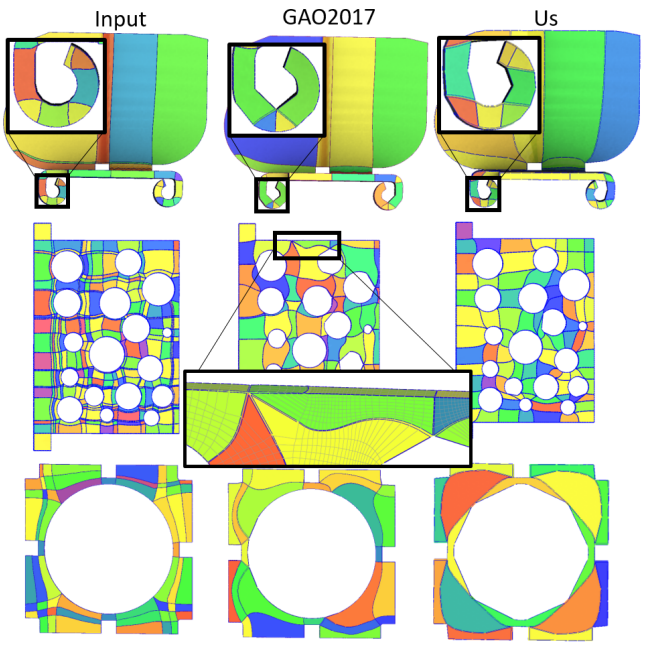

Fig. 14. Visual comparison between our method and the mesh simplification method [Gao et al. 2017]. Quantitative data are reported in Table 2. Our block decomposition creates nicer cells when Gao *et al.* create valency 4 vertices at the boundary of the mesh (middle close-up).

model 4 in Fig. 12 we are able to align the hole with one of the top dents, and this cannot be done by [Cherchi et al. 2016].

Comparison with [Gao et al. 2017]. Another way of computing a minimal block decomposition is to compute a fine hexahedral mesh and to decimate it using a generic mesh simplification algorithm. The state-of-the-art method [Gao et al. 2017] recursively collapses sheets and chords of the mesh until no further simplification is possible. In their algorithm each operation is given a score and the operations are done in a greedy manner. This approach is strictly local and this sequence of simplification does not always yield the coarsest mesh, as illustrated in Fig. 14. It is to be noted that in [Gao et al. 2017] the mesh geometry is recomputed at each step to take into account the simplification operation. In our method the constraints ensure the validity of the polycube map at all time so the mesh geometry can be computed once at the very end of the process. This leads to the considerable speed up reported in Table 2. Finally, the chord simplification introduces new singularities that are not always desirable. For instance in Fig. 14 middle, Gao *et al.* create at the boundary of the mesh valency 4 near valency 2 vertices leading to highly distorted cells.

CONCLUSION

This work provides an important robustness guarantee, but it assumes that a one-to-one $\mathfrak{g}$ is given as input. To achieve a fully robust hexahedral remeshing method based on polycubes, the main remaining difficulty is to solve the flagging problem.

It also opens interesting research opportunities for the general case of hexahedral remeshing based on global parameterizations. Basically, we need to introduce new integer variables corresponding to the (grid preserving) discontinuities in the mapping. The main problem is that the block decomposition is less straightforward to obtain: the parameteric domain becomes an universal covering where the intersection of a plane with the model can be infinite, and tracing it in the geometric domain is an ill-posed problem [Kowalski et al. 2014, 2016].

We have also seen that the combinatorial structure of the hexahedral mesh can be derived from our integer variables. Is it possible to extend this to the general global parameterization case?

REFERENCES

Cecil G. Armstrong, Harold J. Fogg, Christopher M. Tierney, and Trevor T. Robinson. 2015. Common Themes in Multi-block Structured Quad/Hex Mesh Generation. *Procedia Engineering* 124 (Jan. 2015), 70–82. https://doi.org/10.1016/j.proeng.2015.10.123

Marcel Campen, David Bommes, and Leif Kobbelt. 2015. Quantized global parametrization. *ACM Transactions on Graphics* 34, 6 (Oct. 2015), 1–12. https://doi.org/10.1145/2816795.2818140

Long Chen, Gang Xu, Shiyi Wang, Zeyun Shi, and Jin Huang. 2019. Constructing volumetric parameterization based on directed graph simplification of l1 polycube structure from complex shapes. *Computer Methods in Applied Mechanics and Engineering* 351 (July 2019), 422–440. https://doi.org/10.1016/j.cma.2019.01.036

G. Cherchi, P. Alliez, R. Scateni, M. Lyon, and D. Bommes. 2019. Selective Padding for Polycube-Based Hexahedral Meshing. *Computer Graphics Forum* 38, 1 (2019), 580–591. https://doi.org/10.1111/cgf.13593

Gianmarco Cherchi, Marco Livesu, and Riccardo Scateni. 2016. Polycube Simplification for Coarse Layouts of Surfaces and Volumes. *Computer Graphics Forum* 35, 5 (Aug. 2016), 11–20. https://doi.org/10.1111/cgf.12959

CPLEX 2019. IBM ILOG CPLEX Optimization Studio 12.9.0.0. https://www.ibm.com/products/ilog-cplex-optimization-studio

Arbtip Dheeravongkit and Kenji Shimada. 2006. Inverse Adaptation of Hex-dominant Mesh for Large Deformation Finite Element Analysis. In *Geometric Modeling and Processing - GMP 2006*, Myung-Soo Kim and Kenji Shimada (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 189–206.

Xianzhong Fang, Weiwei Xu, Hujun Bao, and Jin Huang. 2016. All-hex meshing using closed-form induced polycube. *ACM Transactions on Graphics* 35, 4 (July 2016), 1–9. https://doi.org/10.1145/2897824.2925957

Xiao-Ming Fu, Chong-Yang Bai, and Yang Liu. 2016. Efficient Volumetric PolyCube-Map Construction. *Computer Graphics Forum* 35, 7 (2016), 97–106. https://doi.org/10.1111/cgf.13007
Xifeng Gao, Zhigang Deng, and Guoning Chen. 2015. Hexahedral mesh re-parameterization from aligned base-complex. *ACM Transactions on Graphics* 34, 4 (July 2015), 142:1–142:10. https://doi.org/10.1145/2766941
Xifeng Gao, Daniele Panozzo, Wenping Wang, Zhigang Deng, and Guoning Chen. 2017. Robust structure simplification for hex re-meshing. *ACM Transactions on Graphics* 36, 6 (Nov. 2017), 1–13. https://doi.org/10.1145/3130800.3130848
James Gregson, Alla Sheffer, and Eugene Zhang. 2011. All-Hex Mesh Generation via Volumetric PolyCube Deformation. *Computer Graphics Forum* 30, 5 (Aug. 2011), 1407–1416. https://doi.org/10.1111/j.1467-8659.2011.02015.x
Hao-Xiang Guo, Xiaohan Liu, Dong-Ming Yan, and Yang Liu. 2020. Cut-enhanced PolyCube-maps for feature-aware all-hex meshing. *ACM Transactions on Graphics* 39, 4 (July 2020). https://doi.org/10.1145/3386569.3392378
Ying He, Hongyu Wang, Chi-Wing Fu, and Hong Qin. 2009. A divide-and-conquer approach for automatic polycube map construction. *Computers & Graphics* 33, 3 (June 2009), 369–380. https://doi.org/10.1016/j.cag.2009.03.024
Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 37, 4 (July 2018), 60:1–60:14. https://doi.org/10.1145/3197517.3201353
Jin Huang, Tengfei Jiang, Zeyun Shi, Yiying Tong, Hujun Bao, and Mathieu Desbrun. 2014. l1Basedd Construction of Polycube Maps from Complex Shapes. *ACM Trans. Graph.* 33, 3 (June 2014), 25:1–25:11. https://doi.org/10.1145/2602141
Felix Kälberer, Matthias Nieser, and Konrad Polthier. 2007. QuadCover - Surface Parameterization using Branched Coverings. *Computer Graphics Forum* 26, 3 (2007), 375–384. https://doi.org/10.1111/j.1467-8659.2007.01060.x
N. Kowalski, F. Ledoux, and P. Frey. 2014. Block-structured Hexahedral Meshes for CAD Models Using 3D Frame Fields. *Procedia Engineering* 82 (Jan. 2014), 59–71. https://doi.org/10.1016/j.proeng.2014.10.373
N. Kowalski, F. Ledoux, and P. Frey. 2016. Smoothness driven frame field generation for hexahedral meshing. *Computer-Aided Design* 72 (March 2016), 65–77. https://doi.org/10.1016/j.cad.2015.06.009
Moritz Lessmann, John Sawyer, and David Knowles. 2018. Methods for Complex Cracked Body Finite Element Assessments. *Procedia Structural Integrity* 13 (2018), 1232 – 1237. https://doi.org/10.1016/j.prostr.2018.12.253 ECF22 - Loading and Environmental effects on Structural Integrity.
Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. 2012. All-Hex Meshing Using Singularity-Restricted Field. *ACM Trans. Graph.* 31, 6, Article Article 177 (Nov. 2012), 11 pages. https://doi.org/10.1145/2366145.2366196
Heng Liu, Paul Zhang, Edward Chien, Justin Solomon, and David Bommes. 2018. Singularity-Constrained Octahedral Fields for Hexahedral Meshing. *ACM Trans. Graph.* 37, 4, Article Article 93 (July 2018), 17 pages. https://doi.org/10.1145/3197517.3201344
Marco Livesu, Alla Sheffer, Nicholas Vining, and Marco Tarini. 2015. Practical Hex-Mesh Optimization via Edge-Cone Rectification. *Transactions on Graphics (Proc. SIGGRAPH 2015)* 34, 4 (2015). https://doi.org/10.1145/2766905
Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. 2013. PolyCut: monotone graph-cuts for PolyCube base-complex construction. *ACM Transactions on Graphics* 32, 6 (Nov. 2013), 1–12. https://doi.org/10.1145/2508363.2508388
Max Lyon, David Bommes, and Leif Kobbelt. 2016. HexEx: robust hexahedral mesh extraction. *ACM Transactions on Graphics* 35, 4 (July 2016), 1–11. https://doi.org/10.1145/2897824.2925976
Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust Field-aligned Global Parametrization. *ACM Trans. Graph.* 33, 4 (July 2014), 135:1–135:14. https://doi.org/10.1145/2601097.2601154
M. Nieser, U. Reitebuch, and K. Polthier. 2011. CubeCover- Parameterization of 3D Volumes. *Computer Graphics Forum* 30, 5 (Aug. 2011), 1397–1406. https://doi.org/10.1111/j.1467-8659.2011.02014.x
Nicolas Ray, Wan Chiu Li, Bruno Lévy, Alla Sheffer, and Pierre Alliez. 2006. Periodic Global Parameterization. *ACM Trans. Graph.* 25, 4 (Oct. 2006), 1460–1485. https://doi.org/10.1145/1183287.1183297
Nicolas Ray, Dmitry Sokolov, and Bruno Levy. 2016. Practical 3D Frame Field Generation. *ACM Trans. Graph.* 35, 6, Article Article 233 (Nov. 2016), 9 pages. https://doi.org/10.1145/2980179.2982408
Dmitry Sokolov and Nicolas Ray. 2015. *Fixing normal constraints for generation of polycubes.* Research Report. LORIA. https://hal.inria.fr/hal-01211408
Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. 2004. PolyCube-Maps. In *ACM SIGGRAPH 2004 Papers (SIGGRAPH '04)*. ACM, New York, NY, USA, 853–860. https://doi.org/10.1145/1186562.1015810 event-place: Los Angeles, California.
Mary Wheeler, Guangri Xue, and Ivan Yotov. 2012. A multipoint flux mixed finite element method on distorted quadrilaterals and hexahedra. *Numer. Math.* 121 (05 2012). https://doi.org/10.1007/s00211-011-0427-7
Hui Zhao, Na Lei, Xuan Li, Peng Zeng, Ke Xu, and Xianfeng Gu. 2018. Robust edge-preserving surface mesh polycube deformation. *Computational Visual Media* 4, 1 (March 2018), 33–42. https://doi.org/10.1007/s41095-017-0100-x
Hui Zhao, Xuan Li, Wencheng Wang, Xiaoling Wang, Shaodong Wang, Na Lei, and Xiangfeng Gu. 2019. Polycube Shape Space. *Computer Graphics Forum* 38, 7 (Oct. 2019), 311–322. https://doi.org/10.1111/cgf.13839

A POSITIVITY OF $\det(Dh)$

In this section gives a proof that [C1] assures the positivity of $\det(Dh)$. We will proceed in two steps, first we will give an mathematical definition for h and then, from it, deduce the necessity of constraint [C1].

A.1 definition of h

In this section, we give a definition of h that is equivalent to the one given in Sec. 4.1.1, but brings another intuition.

We know that each block created by four charts $U_i, U_{i'}, V_j, V_{j'}$ is mapped by $\mathfrak{g}$ to an axis-aligned rectangle (Fig. 15) delimited by $\mathfrak{g}_{1|U_i}, \mathfrak{g}_{1|U_{i'}}, \mathfrak{g}_{2|V_j}$ and $\mathfrak{g}_{2|V_{j'}}$. It is easy to stretch this rectangle to any other axis-aligned rectangle by modifying each axis independently. We can therefore define h as $h(X) := (A_1(\mathfrak{g}_1(X)),\ A_2(\mathfrak{g}_2(X)))$ with $A_1, A_2 : \mathbb{R} \to \mathbb{R}$ being two continuous block-wise affine functions.

By definition, h maps each block defined by four charts $U_i, U_{i'}, V_j, V_{j'}$ to the axis-aligned rectangle delimited by $l_{U_i} + i\varepsilon,\ l_{U_{i'}} + i'\varepsilon,\ l_{V_j} + j\varepsilon,\ l_{V_{j'}} + j'\varepsilon$ for some small $\varepsilon > 0$.

On a block defined by $U_i, U_{i'}, V_j, V_{j'}$, the function $\mathfrak{g}_1$ is bounded by $\mathfrak{g}_{1|U_i}$ and $\mathfrak{g}_{1|U_{i'}}$ in the first dimension. The affine transformation A_1 is defined as follows :

$$\begin{aligned} A_1(\boldsymbol{x}) :&= \frac{l_{U_{i'}} + i'\varepsilon - (l_{U_i} + i\varepsilon)}{\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}}\boldsymbol{x} + b_1 \\ &= a_1\mathfrak{g}_1(x) + b_1 \end{aligned}$$

with b_1 constant chosen such that A_1 maps $\left[\mathfrak{g}_{1|U_i}, \mathfrak{g}_{1|U_{i'}}\right]$ on $\left[l_{U_i} + i\varepsilon, l_{U_{i'}} + i'\varepsilon\right]$. The values of $\mathfrak{g}_{1|U_i}$ and l_{U_i} being defined along the iso-lines, h is continuous at the interface between blocks (but not differentiable).

The affine transformation in the second dimension, A_2, is given by the same construction.

A.2 Computation of $\det(Dh)$

The variation of h are strictly linked to A_1, A_2 and $\mathfrak{g}$, *i.e.* Dh can be expressed as follow :

$$Dh = D\begin{pmatrix} A_1(\mathfrak{g}_1(X)) \\ A_2(\mathfrak{g}_2(X)) \end{pmatrix} = \begin{pmatrix} \frac{\partial h_1}{\partial \mathfrak{g}_1} & \frac{\partial h_1}{\partial \mathfrak{g}_2} \\ \frac{\partial h_2}{\partial \mathfrak{g}_1} & \frac{\partial h_2}{\partial \mathfrak{g}_2} \end{pmatrix} D\mathfrak{g} = \begin{pmatrix} a_1 & 0 \\ 0 & a_2 \end{pmatrix} D\mathfrak{g}$$

meaning that

$$\det(Dh) = a_1 a_2 \det(D\mathfrak{g}).$$

Remembering that $a_1 = \frac{l_{U_{i'}} + i'\varepsilon - (l_{U_i} + i\varepsilon)}{\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}}$, since $\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i} > 0$ and $\mathfrak{g}_{2|V_{j'}} - \mathfrak{g}_{2|V_j} > 0$, for $\det(Dh)$ to be positive, $l_{U_{i'}} + i'\varepsilon - (l_{U_i} + i\varepsilon)$ and $l_{V_{j'}} + j'\varepsilon - (l_{V_j} + j\varepsilon)$ must be of the same sign. This being necessary for all blocks, and by continuity, it means signs must be the same on all the blocks. Being able to consider that $\widetilde{l_{U_i}} = -l_{U_i}$, resp. V_j, for all charts, we deduce constraint [C1], *i.e.* $l_{U_i} \leq l_{U_{i'}}$ and $l_{V_j} \leq l_{V_{j'}}$ on each block.

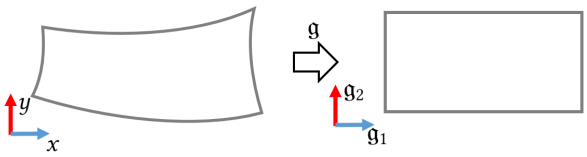

Fig. 15. Each of block of Ω is mapped to a rectangle by $\mathfrak{g}$, which gives a change of coordinate system.

B POSITIVITY OF $\det(DF_L)$ UNDER [C1] AND [C2]

By hypothesis [C1] and [C2]are satisfied by L. Formally it means that:

(1) the coefficient a_1, a_2 are positive (see Appendix A),
(2) for any given edge of the block structure (s,t), $\|h(s)-h(t)\| < |L|\varepsilon \implies \delta(s) = \delta(t)$.

Our goal is to show that, under these assumptions, $\det(DF_L) > 0$. Let's study the determinant of the Jacobian of F_L, using results of Appendix A:

$$\begin{aligned}\det(DF_L) &= \det(Dh - D\delta)\\ &= \det\left(\left[\begin{pmatrix} a_1 & 0\\ 0 & a_2\end{pmatrix} - \begin{pmatrix} \frac{\partial h_1}{\partial \mathfrak{g}_1} & \frac{\partial h_1}{\partial \mathfrak{g}_2}\\ \frac{\partial h_2}{\partial \mathfrak{g}_1} & \frac{\partial h_2}{\partial \mathfrak{g}_2}\end{pmatrix}\right] D\mathfrak{g}\right)\\ &= \det(D\mathfrak{g})\left[\left(a_1 - \frac{\partial \delta_1}{\partial \mathfrak{g}_1}\right)\left(a_2 - \frac{\partial \delta_2}{\partial \mathfrak{g}_2}\right) - \frac{\partial \delta_1}{\partial \mathfrak{g}_2}\frac{\partial \delta_2}{\partial \mathfrak{g}_1}\right]\end{aligned}$$

Recalling that, by construction (see Sec. 4.1.2), $|\frac{\partial \delta_i}{\partial \mathfrak{g}_j}| < |L|\varepsilon$, we can distinguish two cases :

1. $a_1 \geq 1$ and $a_2 \geq 1$:

In this case, ε can be chosen sufficiently small such that $\left(a_1 - \frac{\partial \delta_1}{\partial \mathfrak{g}_1}\right)$ and $\left(a_2 - \frac{\partial \delta_2}{\partial \mathfrak{g}_2}\right)$ are positive, and $\frac{\partial \delta_1}{\partial \mathfrak{g}_2}\frac{\partial \delta_2}{\partial \mathfrak{g}_1}$ is negligible against $a_1 a_2$.

2. $1 > a_1 > 0$ or (inclusive) $1 > a_2 > 0$:

This case is the most difficult. By construction of h, having $1 > a_i > 0$ implies that a_i is of order ε. In the following, using assumption (2), we will show that if a_i is of order ε, then $\frac{\partial \delta_1}{\partial \mathfrak{g}_i} = \frac{\partial \delta_2}{\partial \mathfrak{g}_i} = 0$, which will allow us to concludes that $\det(F_L) > 0$.

First, we must recall that each block of Ω is mapped to a rectangle, with $(\mathfrak{g}_1, \mathfrak{g}_2)$ the new coordinate system, refer to Fig. 15. From this, we notice that along an edge (s,t) of direction $\mathfrak{g}_1$ we have $\|h(s) - h(t)\| = a_1(\mathfrak{g}_1(s) - \mathfrak{g}_1(t))$, which means that $\|h(s) - h(t)\| < |L|\varepsilon$ implies that a_1 is of order ε, and reciprocally as $\mathfrak{g}$ is a constant here.

Subsequently, using the assumption (2), along an edge (s,t) of direction $\mathfrak{g}_1$, $\|h(s) - h(t)\| < |L|\varepsilon \implies \delta(s) = \delta(t)$. Remark that $\delta(s) = \delta(t)$ is equivalent to $\frac{\partial \delta_1}{\partial \mathfrak{g}_1} = \frac{\partial \delta_2}{\partial \mathfrak{g}_1} = 0$ along the edge due to the linearity of δ along (s,t). This means that a_1 of order ε implies $\frac{\partial \delta_1}{\partial \mathfrak{g}_1} = \frac{\partial \delta_2}{\partial \mathfrak{g}_1} = 0$. The same reasoning holds for a_2 and $\mathfrak{g}_2$.

The final problem is to generalize this propriety to the rest of the block (not just along edges). Our construction of δ is bilinear, using the coordinate of each vertex of the block. As the block is a rectangle in h, both opposite edges behave the same, and the behavior of those edge will be interpolated linearly in the block. If $\frac{\partial \delta_i}{\partial \mathfrak{g}_j} = 0$ on both opposite edges, then $\frac{\partial \delta_i}{\partial \mathfrak{g}_j} = 0$ in the whole block. Which means that our result on the edge is also true in the block. ■

What about 3D ? In 3-dimension the proof is strictly equivalent. The only struggle is the development of the det, which is a bit more tedious, but in the end, we find the same results.

C CALCULATION OF $E(h)$

The function h is defined per block, $E(h)$ can be expressed as follows:

$$E(h) = \sum_{b \text{ block of } \Omega} \int_b \|\nabla h_1(X) - \nabla \mathfrak{g}_1(X)\| + \|\nabla h_2(X) - \nabla \mathfrak{g}_2(X)\| dX$$

With b a block of Ω limited by the iso-lines from the charts $U_i, U_{i'}, V_j$ and $V_{j'}$ (cf. fig 9), h_1 is defined such that $h_1(X) = \frac{l_{U_{i'}} - l_{U_i} + (i'-i)\varepsilon}{\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}}\mathfrak{g}_1(X) +$ const (see Appendix A). This give us the following :

$$\begin{aligned}&\int_b \|\nabla h_1(X) - \nabla \mathfrak{g}_1(X)\| dX\\ &= \int_b \left\| \left(\frac{l_{U_{i'}} - l_{U_i} + (i'-i)\varepsilon}{\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}} - 1 \right) \nabla \mathfrak{g}_1(X) \right\| dX\\ &= \left| \frac{l_{U_{i'}} - l_{U_i} + (i'-i)\varepsilon}{\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}} - 1 \right| \int_b \|\nabla \mathfrak{g}_1(X)\| dX\\ &= \left| \frac{l_{U_{i'}} - l_{U_i} + (i'-i)\varepsilon - (\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i})}{\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}} \right| \int_b \|\nabla \mathfrak{g}_1(X)\| dX\\ &= \left| l_{U_{i'}} - l_{U_i} + (i'-i)\varepsilon - (\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}) \right| \frac{\int_b \|\nabla \mathfrak{g}_1(X)\| dX}{|\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}|}\end{aligned}$$

We denote $\alpha_1 = \frac{\int_b \|\nabla \mathfrak{g}_1(X)\| dX}{|\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}|}$ and passing to the limits, it gives :

$$\lim_{\varepsilon \to 0} \int_b \|\nabla h_1(X) - \nabla \mathfrak{g}_1(X)\| dX = \alpha_1 \left| l_{U_{i'}} - l_{U_i} - (\mathfrak{g}_{1|U_{i'}} - \mathfrak{g}_{1|U_i}) \right|$$

We do the same calculation for h_2 with α_2 and it results in equation (4).