Universitat Politècnica de Catalunya
Programa de Doctorat en Matemàtica Aplicada

Barcelona Supercomputing Center
Computer Applications in Science and Engineering Department

Conformal n-dimensional bisection for local refinement of unstructured simplicial meshes

by

Guillem Belda-Ferrín

PhD dissertation
Advisor: Xevi Roca
Co-advisor: Eloi Ruiz-Gironés
Barcelona, September 2022

Dedicat a muns pares i a la Clara

Abstract

Conformal n-dimensional bisection for local refinement of unstructured simplicial meshes

Guillem Belda-Ferrín

In n-dimensional adaptive applications, conformal simplicial meshes must be locally modified. One systematic local modification is to bisect the prescribed simplices while surrounding simplices are bisected to ensure conformity. Although there are many conformal bisection strategies, practitioners prefer the method known as the newest vertex bisection. This method guarantees key advantages for adaptivity whenever the mesh has a structure called reflectivity. Unfortunately, it is not known (i) how to extract a reflection structure from any unstructured conformal mesh for three or more dimensions. Fortunately, a conformal bisection method is suitable for adaptivity if it almost fulfills the newest vertex bisection advantages. These advantages are almost met by an existent multi-stage strategy in three dimensions. However, it is not known (ii) how to perform multi-stage bisection for more than three dimensions.

This thesis aims to demonstrate that n-dimensional conformal bisection is possible for local refinement of unstructured conformal meshes. To this end, it proposes the following contributions. First, it proposes the first 4-dimensional two-stage method, showing that multi-stage bisection is possible beyond three dimensions. Second, following this possibility, the thesis proposes the first n-dimensional multi-stage method, and thus, it answers question (ii). Third, it guarantees the first 3-dimensional method that features the newest vertex bisection advantages, showing that these advantages are possible beyond two dimensions. Fourth, extending this possibility, the thesis guarantees the first n-dimensional marking method that extracts a reflection structure from any unstructured conformal mesh, and thus, it answers question (i). This answer proves that local refinement with the newest vertex bisection is possible in any dimension. Fifth, this thesis shows that the proposed multi-stage method almost fulfills the advantages of the newest vertex bisection. Finally, to visualize four-dimensional meshes, it proposes a simple tool to slice pentatopic meshes.

In conclusion, this thesis demonstrates that conformal bisection is possible for local refinement in two or more dimensions. To this end, it proposes two novel methods for unstructured conformal meshes, methods that will enable adaptive applications on n-dimensional complex geometry.

Acknowledgments

This thesis is the result of a team of different people's shared effort, dedication, endurance, and ideas. Thus, for me, it is necessary to thank everyone who made this dissertation possible.

I want to thank Xevi Roca for his guidance and advice. Thanks for allowing me to be a researcher and teaching me across all categories, professionally and personally. Especialment per totes les "*apretades*" i sobretot per fer-me veure que, fem el que fem, hem de disfrutar del camí. I also would like to thank Eloi Ruiz Gironés, my second advisor *in light* and *partner in breakfast*. Without his positive energy, this adventure would not have been the same. Gràcies per tants bons moments, riures, suport i "*trolling*". Working with Xevi and Eloi was a true privilege. De veritat, moltes gràcies als dos.

Thank all the members of the meshing group for these years. I am grateful to Abel Gargallo for the shared ideas and good vibes. I am also grateful to Guillermo Aparicio and Albert Jiménez. We supported, encouraged, and cared for each other during these years, staying together during the tough times of the doctorate. Keep that path; you are doing well. Now it's your bureaucracy turn.

I appreciate very much the possibility of doing my thesis in the Computer Applications in Science and Engineering (CASE) research framework at Barcelona Supercomputing Center (BSC). I am grateful to thank the support and friendship of all the workmates at CASE. Especialment a en Gerard Guillamet, per tots els riures, ànims i "*has d'anar de cara barraca*" de la recta final. També a l'Adrià Quintanas, l'Albert Coca i en Roger Pastor per formar el grup políticament més incorrecte del departament. Gràcies.

Thanks a lot to the Facultat de Matemàtiques i Estadística (FME). To Sonia Fernandez for introducing me to applied mathematics and for communicating to me the possibility of working at BSC. To Jaume Franch for opening to me the research door of applied mathematics. To all the Ph.D. crew of DMA, we have shared coffees, teas, and beers. Especially to Mar Giralt for sharing with me this last year. Què fariem sense el nostre "*gossipeo*"! Finally, to Daniel Aldeguer, Jordi Pizarro, and "Paupi", the most dangerous squad of lovely people that FME ever had.

I want to show my gratitude to the tribunal members and the referees. It is difficult to add new entries to a crowded agenda with the current situation, but they made it possible. Especially to Pep Sarrate for being part of this thesis committee from the beginning.

També donar-li les gràcies a la família Mateo Campo. Per aquesta aventura i per les que vindran. También a mis amigos Anna, Esther y López, ese pequeño grupo de gente que me acompaña desde los cuatro años y que tengo la suerte de poder llamar mis amigas. A Sergi Picart, porque las penas compartidas son menos penas.

Finalment, donar les gràcies a la meva família pel suport i l'ajuda durant aquests 31 anys. A la Nieves i a en Quim, els meus pares, per ser els pilars sobre els quals vaig començar a fer la meva vida i als quals hi torno quan necessito retrobar-me amb l'esència. A Adrián, mi hermano, por escuchar, guiar y relativizar mi yo y mi porqué. I finalment a la Clara, la meva companya de vida, per estar al meu costat fent equip en tot moment. Us estimo molt.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation and background

In many applications, computational scientists and engineers need to numerically solve problems formulated in more than three dimensions, problems such as partial differential equations (PDEs) and combinatorial fixed points, problems that model physical and economic phenomena. In these problems, the dimensionality might be relatively small or potentially large. It is relatively small in problems such as 4D general relativity equations, 4D space-time discretizations of unsteady 3D phenomena, and 6D Boltzmann equations. It is potentially large in problems such as the Black-Scholes equation with as many dimensions as financial assets, the Schrödinger equation with as many dimensions as three times the number of quantum particles, and combinatorial fixed point models featuring up to thousands of dimensions. Unfortunately, the cost of solving these problems grows exponentially with dimensionality.

To mitigate this exponential growth, practitioners can use adaptive mesh refinement to reduce the element count and, thus, the computational cost. Specifically, adaptive mesh refinement uses finer elements where the solution presents larger variations and coarser elements where the solution presents smaller variations.

In adaptive n-dimensional refinement, conformal simplicial meshes must be locally modified. One systematic modification for arbitrary dimensions is to bisect a set of selected simplices. This operation splits each simplex by introducing a new vertex on a previously selected refinement edge. Then, this new vertex is connected to the original vertices to define two new simplices. To ensure that the mesh is still conformal, the

bisection has to select additional refinement edges on a surrounding conformal closure. This edge selection is commonly based on choosing either the longest edge (Rivara, 1984, 1991; Plaza and Carey, 2000; Plaza and Rivara, 2003) or the newest vertex (Mitchell, 1991; Kossaczký, 1994; Maubach, 1995, 1996b; Traxler, 1997). Although both edge selections are well-suited for adaption, newest vertex bisection has been preferred in applications where it is possible to start with a reflected mesh (Maubach, 1995; Traxler, 1997; Stevenson, 2008; Alkämper et al., 2018).

This preference is so since on reflected meshes newest vertex bisection guarantees key advantages for n-simplicial adaption. The first advantage is that if the initial mesh is conformal, the refined mesh is also conformal (Stevenson, 2008) (conformity). Second, local refinement of a set of simplices terminates in finite time (Stevenson, 2008) (finiteness). Third, successive refinement leads to a fair number of simplex similarity classes (Maubach, 1996a; Traxler, 1997; Arnold et al., 2000; Stevenson, 2008). Thus, the minimum mesh quality is bounded (stability). Finally, it needs a fair number of additional bisections to complete the conformal closure (Stevenson, 2008) (locality).

Unfortunately, practitioners only know how to exploit the advantages of pure newest vertex bisection on meshes generated using the Coxeter-Freudenthal-Kuhn algorithm (Coxeter, 1934; Freudenthal, 1942; Kuhn, 1960) or meshes where all the edges have an even number of incident tetrahedra (Maubach, 1996b; Traxler, 1997). For complex geometry, newest vertex bisection is only used in two dimensions (Mitchell, 1991). For more dimensions, there is not any known method to extract a reflection structure from an arbitrary unstructured conformal mesh (Maubach, 1995; Traxler, 1997; Stevenson, 2008; Alkämper et al., 2018).

For unstructured conformal meshes, there are pre-processing methods for n-dimensional meshes that allow conformal finite termination of posterior newest vertex bisection (Stevenson, 2008; Alkämper et al., 2018). However, these methods do not fulfill simultaneously the similarity and the locality properties. The number of similarity classes for the method in Stevenson (2008) is not favorably bounded since the pre-process starts by splitting all simplices into $(n+1)!/2$ simplices. Thus, the method can lead to $(n+1)!/2$ times more similarity classes for each simplex than pure newest vertex bisection. On the contrary, the bound on the cost of the conformal closure for the method in Alkämper et al. (2018) might be worst than in Stevenson (2008) since the pre-process leads to weakly reflected meshes. As stated in Alkämper et al.

(2018), this weak condition might not satisfy the strong reflection condition required for locality Stevenson (2008), a strong condition that is possibly more restrictive than being a weakly reflected mesh.

Fortunately, a conformal and finite bisection method is adequate for adaptive analysis if it almost fulfills the similarity and locality properties. To almost fulfill these conditions, one can use an existent multi-stage method (Arnold et al., 2000). These methods feature a first stage performing a specific-purpose bisection for marked simplices. This marked bisection enforces that after a few initial steps, one can switch independently on each element to another stage featuring Maubach's newest vertex bisection (Maubach, 1995). The number of initial bisection steps is comparable with the spatial dimension. Hence, these steps are responsible for an initial slight increase in both the total number of similarity classes and the cost of the conformal closure. Taking into account this initial increase and since the first stage fulfills the sufficient conformity and reflection conditions stated in Stevenson (2008), the whole method is finite and conformal while almost fulfills the similarity and locality properties. However, these multi-stage methods are specifically devised for 3D by Arnold et al. (2000). Accordingly, the question of whether there is a practical n-dimensional multi-stage bisection method for unstructured conformal meshes is still open.

1.2 Research opportunity and questions

The previous overview *identifies a research opportunity to enable mesh adaptivity in arbitrary dimensions*. Although there are methods for conformal bisection of unstructured conformal simplicial meshes in arbitrary dimensions, *there is no known method that guarantees (almost guarantees) the advantages for local refinement of newest vertex bisection on unstructured conformal meshes in more than two (three) dimensions*.

The overview also identifies the following key research questions for conformal bisection of unstructured conformal simplicial meshes in arbitrary dimensions:

(Q1) *Can a reflection structure be extracted from any mesh?* Answering this question will enable the advantages of the newest vertex bisection for local refinement of complex geometry. This is an open question for three (Arnold et al., 2000) or more dimensions (Mitchell, 1991, 2017).

(Q2) *Is it possible to perform multi-stage bisection in any mesh?* Answering this question will provide a method that almost fulfills the advantages of the newest vertex bisection for local refinement of complex geometry. This is an open question for four or more dimensions (Arnold et al., 2000).

Potentially, answering question (Q2) is easier than answering question (Q1). Because the newest vertex bisection can be understood as a multi-stage method with a first stage featuring zero refinements, we have that answering question (Q1) also answers question (Q2). On the contrary, because the first stages of a multi-stage bisection method might not be equivalent to the newest vertex bisection, answering question (Q2) does not answer question (Q1).

1.3 Aim and objectives

This thesis aims to demonstrate conformal bisection methods in arbitrary dimensions for local refinement of unstructured conformal simplicial meshes. To this end, this thesis develops the following *objectives*:

(O1) *Proposing a multi-stage bisection method in four dimensions*, Appendix D.

(O2) *Proposing a multi-stage bisection method in arbitrary dimensions*, Chapter 3.

(O3) *Enabling and guaranteeing newest vertex bisection advantages in three dimensions*, Chapter 4.

(O4) *Enabling and guaranteeing newest vertex bisection advantages in arbitrary dimensions*, Chapter 5.

(O5) *Demonstrating suitability of multi-stage bisection method for local refinement in arbitrary dimensions*, Chapter 6.

The aim of this thesis addresses the research opportunity. Moreover, the objectives address the research questions. Objectives (O1) and (O2) empirically address the research question (Q2). Objectives (O3) and (O4) formally address the research question (Q1). Objective (O5) proposes a formal outline for the research question (Q2).

1.4 Scope

This thesis is focused on conformal bisection methods for local refinement on unstructured conformal simplicial meshes. Specifically, this thesis is focused on marked bisection and the newest vertex bisection methods. Following, we justify this scope, yet there are alternative choices. First, refinement with bisection is preferred because it features a nested structure that can be exploited, *e.g.*, by fast multigrid solvers. Second, bisection conformity is favored because it allows bisecting conformal and non-conformal meshes. Third, unstructured simplicial meshes are chosen because they feature geometric flexibility, and there are automatic mesh generators for complex (simple) geometries in two and three (arbitrary) dimensions. Fourth, marked bisection and the newest vertex bisection methods are preferred because they rely on existing theoretical results guaranteeing advantages for adaptivity.

1.5 Methodology

To gradually meet the aim of this thesis, the research methodology increases the potential difficulty of the objectives. Because multi-stage methods are more flexible, the thesis starts with objectives (O1) and (O2). These objectives propose multi-stage methods, the first demonstrates a particular method for four dimensions, and the second demonstrates a generic method in arbitrary dimensions. Then, encouraged by these multi-stage bisection results and deriving new sufficient conditions for newest vertex bisection, the objectives (O3) and (O4) are focused on the newest vertex bisection. Specifically, they guarantee the advantages of the newest vertex bisection in three (O3) and arbitrary dimensions (O4). Finally, using the reflectivity theory developed in (O4), outlining new approaches to ensure conformity, and estimating the number of similarity classes, objective (O5) shows the suitability of marked bisection in arbitrary dimensions for local refinement.

The methodology approaches are based on computer implementations, runtime checks, and formal proofs. First, the computer implementations of the proposed methods allow showing empirical evidence of the advantages for local refinement. Second, the computer implementation checks mesh invariants at runtime, invariants such as reflectivity and conformity, invariants that validate the obtained results. Finally, the formal proofs guarantee the properties of the proposed methods. These theoretical proofs rely on existent theory, new conjectures, and new theoretical results.

1.6 Contributions and novelty

This thesis demonstrates conformal bisection for local refinement of unstructured meshes in arbitrary dimensions. To this end, *this thesis proposes novel methods*:

(C1) *The first multi-stage bisection in four dimensions*, Appendix D. This contribution empirically answers question (Q2) in four dimensions. It corresponds to the peer-reviewed conference paper Belda-Ferrín et al. (2019).

(C2) *The first multi-stage bisection in arbitrary dimensions*, Chapter 3. This contribution empirically answers question (Q2) in arbitrary dimensions. It corresponds to the submitted journal paper Belda-Ferrín et al. (2022).

(C3) *The first guaranteed marked bisection behaving as the newest vertex bisection in three dimensions*, Chapter 4. This contribution empirically and formally answers question (Q1) in three dimensions. It corresponds to the peer-reviewed conference paper Belda-Ferrín et al. (2021).

(C4) *The first guaranteed newest vertex bisection in arbitrary dimensions*, Chapter 5. This contribution empirically and formally answers question (Q1) in arbitrary dimensions. It corresponds to the journal paper in preparation Belda-Ferrín et al. (2022a).

(C5) *An outline to guarantee multi-stage bisection for local refinement in arbitrary dimensions*, Chapter 6. This contribution addresses question (Q2). It checks the question in two and three dimensions, and it outlines a formal answer in arbitrary dimensions. This contribution corresponds to the journal paper in preparation Belda-Ferrín et al. (2022b).

(C6) *A slicing tool to visualize four-dimensional pentatopic meshes*, Appendix B. This contribution corresponds to the research note Belda-Ferrín et al. (2019).

Reflectivity and conformity are sufficient conditions to use the newest vertex bisection method on unstructured conformal meshes. Accordingly, the *central findings of this thesis provide sufficient conditions to ensure reflectivity and conformity. To ensure reflectivity, this thesis proves that it is sufficient to sort the mesh vertices with a strict and total order*, Chapter 5. *To ensure conformity, this thesis promotes checking conformity separately on faces determined by mesh simplices and on faces generated by bisection*, Chapter 6.

1.7 Layout

To develop the previous contributions, the thesis starts with the preliminaries and definitions, contents that are common to the different contributions. Then, the proposed contributions are successively presented. Finally, the appendices include the detail of non-central methods and results. The final appendix appends a peer-reviewed conference paper corresponding to contribution (C1). Following, we summarize the contents of the contribution chapters.

In Chapter 3, we present an n-dimensional marked bisection method for unstructured conformal meshes. We devise the method for local refinement in adaptive n-dimensional applications. To this end, we propose a mesh marking pre-process and three marked bisection stages. The pre-process marks the initial mesh conformingly. Then, in the first $n-1$ bisections, the method accumulates in reverse order a list of new vertices. In the second stage, the n-th bisection, the method uses the reversed list to cast the bisected simplices as reflected simplices, a simplex type suitable for newest vertex bisection. In the final stage, beyond the n-th bisection, the method switches to newest vertex bisection. To allow this switch, after the second stage, we check that under uniform bisection the mesh simplices are conformal and reflected. These conditions are sufficient to use newest vertex bisection, a bisection scheme guaranteeing key advantages for local refinement. Finally, the results show that the proposed bisection is well-suited for local refinement of unstructured conformal meshes.

In Chapter 4, we propose a new method to mark for bisection the edges of an arbitrary three-dimensional unstructured conformal mesh. For these meshes, the approach conformingly marks all the tetrahedra with coplanar edge marks. To this end, the method needs three key ingredients. First, we propose a specific edge ordering. Second, marking with this ordering, we guarantee that the mesh becomes conformingly marked. Third, we also ensure that all the marks are coplanar in each tetrahedron. To demonstrate the marking method, we implement an existent marked bisection approach. Using this implementation, we mark and then locally refine three-dimensional unstructured conformal meshes. We conclude that the resulting marked bisection features an optimal bound of 36 similarity classes per tetrahedron.

In Chapter 5, we propose a new method to mark for bisection the edges of an arbitrary n-dimensional unstructured conformal mesh. This marking method is devised for local refinement in adaptive n-dimensional applications. To this end, we

consider three main ingredients. First, we sort the simplex vertices with a strict and total order, and we mark all the mesh elements as Maubach simplices with a tag equal to n. Second, for any n-dimensional unstructured conformal mesh, we guarantee that the marking procedure leads to a reflected mesh. Third, we guarantee that the resulting meshes are strongly compatible, and thus, they are suitable for posterior local refinement with the newest vertex bisection. To illustrate the local refinement application, we equip an n-dimensional refine to conformity marked bisection implementation with our marking method. With this implementation, we locally refine unstructured conformal meshes of different dimensions. We also illustrate the stability and locality of the resulting meshes. We conclude that the resulting marked bisection enables the conformity, finiteness, stability, and locality properties of the newest vertex bisection on complex n-dimensional geometry.

In Chapter 6, we show that the resulting meshes are conformal, the algorithm finishes in a finite number of steps, and that the meshes asymptotically feature locality. To this end, we illustrate and justify that it is sufficient to prove that after n uniform multi-stage bisections the resulting mesh is conformal and reflected. Finally, we estimate that the number of generated similarity classes is slightly greater than with the newest vertex bisection. In conclusion, because the proposed method is conformal, finite, asymptotically local, and features a fair similarity bound, it can be used for local refinement of unstructured conformal meshes.

In Appendix B, we propose a simple tool to visualize 4D unstructured pentatopic meshes. We present a method that slices unstructured 4D pentatopic meshes (fields) with an arbitrary 3D hyperplane and obtains a conformal 3D unstructured tetrahedral representation of the mesh (field) slice ready to explore with standard 3D visualization tools. The results show that the method is suitable to visually explore 4D unstructured meshes. This capability has facilitated devising our 4D bisection method, and thus, we think it might be useful when devising new 4D meshing methods. Furthermore, it allows visualizing 4D scalar fields, which is a crucial feature for our space-time applications.

In Appendix D, we present a conformal bisection procedure for local refinement of 4D unstructured simplicial meshes with bounded minimum shape quality. Specifically, we propose a recursive refine to conformity procedure in two stages, based on marking bisection edges on different priority levels and defining specific refinement templates. Two successive applications of the first stage ensure that any 4D

unstructured mesh can be conformingly refined. In the second stage, the successive refinements lead to a cycle in the number of generated similarity classes and thus, we can ensure a bound over the minimum shape quality. In the examples, we check that after successive refinement the mesh quality does not degenerate. Moreover, we refine a 4D unstructured mesh and a space-time mesh (3D+1D) representation of a moving object.

Chapter 2

Preliminaries and definitions

Before proposing the new methods and results of this thesis, we introduce the common notation, concepts, theory, and methods. First, we present existent content in the literature related to simplicial meshes, conformity, and bisection methods. Second, we introduce novel concepts and methods, contents that are required by the contribution chapters.

2.1 Preliminaries

We proceed to introduce the necessary notation and concepts. Specifically, we introduce the preliminaries related simplicial meshes, conformity, and bisection methods.

2.1.1 Simplicial meshes, conformity, and bisection

A *simplex* is the convex hull of $n+1$ points $p_0, \ldots, p_n \in \mathbb{R}^n$ that do not lie in the same hyperplane. We denote a simplex as $\sigma = conv(p_0, \ldots, p_n)$. We identify each point p_i with a unique integer identifier v_i that we refer as *vertex*. Thus, a simplex is composed of $n+1$ vertices and we denote it as $\sigma = (v_0, \ldots, v_n)$ where v_i is the identifier of point p_i. We have an application Π that maps each identifier v_i to the corresponding point p_i.

Given a simplex σ, a *k-entity* is a sub-simplex composed of $k+1$ vertices of σ, for $0 \leq k \leq n-1$. We say that a 1-entity is an *edge* and an $(n-1)$-entity is a *face*.

The number of k-entities contained in a simplex σ is

$$\binom{n+1}{k+1}.$$

Particularly, the number of edges and faces of σ is

$$\binom{n+1}{2} = \frac{n(n+1)}{2}, \quad \binom{n+1}{n} = n,$$

respectively. We associate each face of a simplex σ to its opposite vertex in σ. Specifically, the opposite face to v_i is

$$\kappa_i = (v_0, v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n).$$

We say that two simplices σ_1 and σ_2 are *neighbors* if they share a face.

We define a *mesh*, $\mathcal{T}$, associated to an open set $\Omega \in \mathbb{R}^n$ as a finite collection of mutually disjointed simplices such that

$$\overline{\Omega} = \bigcup_{\sigma \in \mathcal{T}} \sigma.$$

A simplicial mesh is *conformal* if the following two conditions are satisfied:

(C1) For any $\sigma \in \mathcal{T}$, $\sigma \cap \partial\Omega$ is the union of entities of σ.

(C2) For any $\sigma_1, \sigma_2 \in \mathcal{T}$, either $\sigma_1 \cap \sigma_2$ is empty, or a k-entity of $\mathcal{T}$ with $0 \leq k \leq n-1$.

In reference Stevenson (2008), the author states a definition for conformal meshes slightly different from the standard one. Specifically, he considers that a simplicial mesh is *conformal* if the following two conditions are satisfied:

(C1) For any $\sigma \in \mathcal{T}$, $\sigma \cap \partial\Omega$ is the union of entities of σ.

(C3) Each $x \in \sigma_1 \cap \sigma_2$, with $\sigma_1, \sigma_2 \in \mathcal{T}$, for which any open ball $B \ni x$, any $y \in \sigma_1 \cap B \cap \Omega$, $y' \in \sigma_2 \cap B \cap \Omega$ are connected by a path through $B \cap \Omega$, lies on a joint k-entity of σ_1 and σ_2.

When Ω nowhere lies simultaneously on both sides of an $(n-1)$- dimensional part of its boundary, (C3) is equivalent to the standard definition (C2) of conformity. If, in addition, $\partial\Omega$ is everywhere $(n-1)$-dimensional, i.e., if $\Omega = \text{int}(\overline{\Omega})$, then (C3) implies (C1), see Remark 3.1 of Stevenson (2008).

With that definition, Stevenson proves that if a mesh $\mathcal{T}$ fulfills the condition (C3) then, it is enough to check the conformity of $\mathcal{T}$ only through faces. We rewrite its theorem as follows:

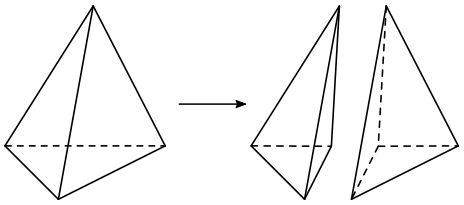


Figure 2.1: Bisection of a tetrahedron into two tetrahedra.

Theorem 2.1 (Theorem 3.2 of Stevenson (2008))**.** *A mesh $\mathcal{T}$ of Ω satisfies (C3) if and only if any $\sigma_1, \sigma_2 \in \mathcal{T}$, for which $\sigma_1 \cap \sigma_2 \cap \Omega$ contains a point interior to a face of σ_1, or σ_2 are neighbors.*

Specifically, if a mesh $\mathcal{T}$ is conformal by faces, then Theorem 2.1 ensures automatically that $\mathcal{T}$ is conformal by all the k-entities, where $1 \leq k \leq n-2$. Thus, Theorem 2.1 ensures that checking the conformity of the faces automatically checks the conformity of all the entities of the mesh.

We define the *bisection* of a simplex as the operation that splits a simplex by introducing a new vertex on the selected refinement edge. Then, the vertices not lying on this refinement edge are connected to the new vertex. These connections determine two new simplices. Figure 2.1 shows the bisection of a tetrahedron.

2.1.2 Newest vertex bisection

We introduce tagged-simplex bisection, see Maubach (1995), and some definitions that explain the sufficient conditions to use newest vertex bisection, see Stevenson (2008).

We define a *tagged simplex* as a simplex $\sigma = (v_0, v_1, \ldots, v_n)$ equipped with an integer called *tag*, $d \in \{1, \ldots, n\}$. We denote it as $\sigma = (v_0, v_1, \ldots, v_n)_d$. Maubach proposes a bisection step for tagged simplices, see Algorithm 2.1. The input is a tagged simplex σ, and the output are two tagged simplices σ_1 and σ_2.

Let $\sigma = (v_0, \ldots, v_n)_d$ the input tagged simplex. First, we define the new tag of the children as d', see Line 3. Then, we create the new vertex z, which is the midpoint of the edge (v_0, v_d), see Line 4. Then, we define the two tagged children as $\sigma_1 = (v_0, v_1, \ldots, v_{d-1}, z, v_{d+1}, \ldots, v_n)_{d'}$ and $\sigma_2 = (v_1, v_2, \ldots, v_d, z, v_{d+1}, \ldots, v_n)_{d'}$, see Lines 5 and 6, respectively. Finally, we return the two tagged simplices σ_1 and σ_2, see Line 7.

Algorithm 2.1 Maubach's bisection of a n-simplex.

input: `TaggedSimplex` σ
output: `TaggedSimplex` σ_1, `TaggedSimplex` σ_2

function `bisectTaggedSimplex`(σ)
  $(v_0, \ldots, v_n)_d = \sigma$
  Set $d' = \begin{cases} d-1, & d > 1 \\ n, & d = 1 \end{cases}$
  Create the new vertex $z = \frac{1}{2}(v_0 + v_d)$
  $\sigma_1 = (v_0, v_1, \ldots, v_{d-1}, z, v_{d+1}, \ldots, v_n)_{d'}$.
  $\sigma_2 = (v_1, v_2, \ldots, v_d, z, v_{d+1}, \ldots, v_n)_{d'}$.
  return μ_1, μ_2
end function

According to Stevenson (2008), we say that two neighboring tagged simplices $\sigma_1 = (v_0, \ldots, v_n)_{d_1}$ and $\sigma_2 = (w_0, \ldots, w_n)_{d_2}$ with $d_1 = d_2$ are *reflected neighbors* if the order of vertices of the shared face is the same. That is, there are two sequences $i_0 < i_1 < \ldots < i_{n-1}$ and $j_0 < j_1 < \ldots < j_{n-1}$ such that $v_{i_k} = w_{j_k}$, for $k = 0, \ldots, n-1$ and $i_k, j_k \in \mathbb{N}$. In reference Traxler (1997), the author defines a *reflected mesh* as a mesh such that all adjacent pairs of simplices are reflected neighbors.

In reference Alkämper et al. (2018), the authors introduce some definitions about the compatibility of a face and of a mesh. A face $\kappa = \sigma_1 \cap \sigma_2$ of the mesh $\mathcal{T}$ is called a *strongly compatible face* if σ_1 and σ_2 are reflected neighbors, or their children adjacent to κ are reflected neighbors. A mesh $\mathcal{T}$ is a *strongly compatible mesh* if all faces in $\mathcal{T}$ are strongly compatible.

Remark 2.1 (Strong compatibility condition)**.** Note that to have a strongly compatible mesh, it is sufficient to have a conformal and reflected mesh.

Remark 2.2 (Newest vertex bisection conditions)**.** If the initial mesh is strongly compatible, then we can use newest vertex bisection and have the following properties:

- **Conformity and finiteness.** Local refinement with newest vertex bisection generates a conformal mesh and terminates in a finite number of steps, see Theorem 5.1 of Maubach (1995) and Theorem 5.1 of Stevenson (2008).

- **Stability.** Successive refinement of newest vertex bisection leads at most to $M_n = nn!2^{n-2}$ similarity classes, see Theorem 4.5 of Arnold et al. (2000). Thus, the minimum mesh quality of the refined mesh is bounded.

- **Locality.** Local refinement with newest vertex bisection needs a fair number of additional bisections to complete the conforming closure, see Theorem 2.4 of Binev et al. (2004) and Theorem 6.1 of Stevenson (2008). The number of simplices of the refined mesh is bounded, up to a constant, by the sum of refined simplices of the initial mesh. That is, let $\mathcal{T}_0$ be a conformal simplicial mesh, $\mathcal{T}_n$ be the mesh obtained after the n-th local refinement iteration, $\mathcal{M}_k$ be the set of elements marked to be refined at the k-th iteration of local refine, and $\mathcal{M} = \mathcal{M}_0 \cup \cdots \cup \mathcal{M}_{n-1}$ be the union of all the marked sets. Then, there exists a constant $C > 0$ such that

$$\#(\mathcal{T}_n) - \#(\mathcal{T}_0) \leq C\#(\mathcal{M}) \tag{2.1}$$

  is hold, where $\#(\cdot)$ denotes the number of simplices. Note that the constant C only depends on the initial mesh, $\mathcal{T}_0$, and it does not depend on the number of refinement iterations.

2.1.3 Marked bisection

In reference Arnold et al. (2000), the authors presented a marked bisection algorithm for unstructured conformal tetrahedral meshes that ensures locally refined conformal meshes and quality stability. The *refinement edge* is the edge of σ to be bisected. Since an edge is shared by $n-1$ faces of the simplex, the faces that contain the refinement edge are the *refinement faces* of σ. The remaining two faces are defined as *non-refinement faces*.

Now, we can introduce the definition of *marked simplex*, which is a modification of the one detailed by Arnold et al. (2000). Herein, a marked simplex is a simplex σ equipped with a data structure that tells us how to refine it and its descendants.

A mesh is *marked* if all its simplices are marked. A marked conformal mesh is *conformingly-marked* if each entity has a unique refinement edge. That is, an entity shared by two simplices has the same refinement edge from both sides. Accordingly, shared entities are bisected in the same manner from different simplices.

To perform the bisection process, we adapt to the n-dimensional case the recursive refine-to-conformity scheme proposed by Arnold et al. (2000). The marked bisection method, Algorithm 2.2, starts by marking the initial unstructured conformal mesh and then applies a local refinement procedure to a set of simplices of the marked mesh. To do it so, we need to specify a conformal marking procedure for simplices to

Algorithm 2.2 Refining a subset of a mesh.

input: `Mesh` $\mathcal{T}$, `SimplicesSet` $\mathcal{M} \subset \mathcal{T}$
output: `ConformalMarkedMesh` $\mathcal{T}_2$

```
1: function refineMesh(T, M)
2:     T_1 = markMesh(T)
3:     T_2 = localRefine(T_1, M)
4:     return T_2
5: end function
```

Algorithm 2.3 Local refinement of a marked mesh.

input: `ConformalMarkedMesh` $\mathcal{T}$ and `SimplicesSet` $\mathcal{M} \subset \mathcal{T}$
output: `ConformalMarkedMesh` $\mathcal{T}'$

```
1: function localRefine(T, M)
2:     T̄ = bisectSimplices(T, M)
3:     T' = refineToConformity(T̄)
4:     T_0 = renumberMesh(T')
5:     return T_0
6: end function
```

Algorithm 2.4 Refine-to-conformity a marked mesh.

input: `MarkedMesh` $\mathcal{T}$
output: `MarkedMesh` $\mathcal{T}'$ without hanging vertices

```
1: function refineToConformity(T)
2:     M = getNonConformalSimplices(T)
3:     if M ≠ ∅ then
4:         T̄ = bisectSimplices(T, M)
5:         T' = refineToConformity(T̄)
6:     else
7:         T' = T
8:     end if
9:     return T'
10: end function
```

obtain a marked mesh $\mathcal{T}'$. Using this marked mesh, the local refinement procedure, Algorithm 2.3, first refines a set of simplices, then calls a recursive refine-to-conformity strategy, and finally renumbers the mesh, see C.3. The refine-to-conformity strategy, Algorithm 2.4, terminates when successive bisection leads to a conformal mesh. Both algorithms use marked bisection to refine a set of elements, see Algorithm 2.5.

Remark 2.3 (Declaring new bisection methods)**.** In the following chapters, we use the previous bisection process to declare new bisection methods. To this end, for each

Algorithm 2.5 Bisect a set of simplices.

input: `MarkedMesh` $\mathcal{T}$, `SimplicesSet` $\mathcal{M}$
output: `MarkedMesh` $\mathcal{T}_1$

function `bisectSimplices`($\mathcal{T}$, $\mathcal{M}$)
  $\mathcal{T}_1 = \emptyset$
  for $\rho \in \mathcal{T}$ **do**
    if $\rho \in \mathcal{M}$ **then**
      $\rho_1, \rho_2 =$ `bisectSimplex`(ρ)
      $\mathcal{T}_1 = \mathcal{T}_1 \cup \rho_1$
      $\mathcal{T}_1 = \mathcal{T}_1 \cup \rho_2$
    else
      $\mathcal{T}_1 = \mathcal{T}_1 \cup \rho$
    end if
  end for
  return $\mathcal{T}_1$
end function

method, we specify a mesh preprocess to mark the initial unstructured conformal mesh, see Algorithm 2.2, and a set of simplex bisection rules to refine each element type, see Algorithm 2.5.

In reference Arnold et al. (2000), the authors introduce the following notation to indicate that a mesh has been refined uniformly k times. Let $\mathcal{Q}_0 = \mathcal{T}$ and

$$\mathcal{Q}_k = \texttt{bisectSimplices}(\mathcal{Q}_{k-1}, \mathcal{Q}_{k-1})$$

be the obtained mesh after performing k uniform refinements. The mesh $\mathcal{Q}_i$ is composed of $\#(\mathcal{Q}_i) = 2^i \#(\mathcal{T})$ simplices. Analogously, when we refine uniformly a simplex σ, we denote $\mathcal{Q}_0^\sigma = \sigma$, and

$$\mathcal{Q}_k^\sigma = \texttt{bisectSimplices}(\mathcal{Q}_{k-1}^\sigma, \mathcal{Q}_{k-1}^\sigma)$$

be the obtained mesh after performing k uniform refinements. Since we are performing uniform refinements, the mesh $\mathcal{Q}_i^\sigma$ is composed of $\#(\mathcal{Q}_i^\sigma) = 2^i$ simplices.

2.2 Definitions in this thesis

Before we detail our marked bisection method, we introduce: the notion of multi-id to provide a unique identifier to the mid-vertices; the search of simplices that define

Algorithm 2.6 Generation of a new multi-id.

input: `Multi-Id` $\mathbf{v}_1$, `Multi-Id` $\mathbf{v}_2$
output: `Multi-Id` $\mathbf{v}$

1: **function** `midVertex`$(\mathbf{v}_1, \mathbf{v}_2)$
2: $[v_{1,1}, \ldots, v_{1,k_1}] = \mathbf{v}_1$
3: $[v_{2,1}, \ldots, v_{2,k_2}] = \mathbf{v}_2$
4: $[v_{i_1}, \ldots, v_{i_{k_1+k_2}}] =$ `sort`$([v_{1,1}, \ldots, v_{1,k_1}, v_{2,1}, \ldots, v_{2,k_2}])$
5: $\mathbf{v} = [v_{i_1}, \ldots, v_{i_{k_1+k_2}}]$
6: **return v**
7: **end function**

a non-conformal configuration; the selection of the bisection edge in a consistent manner; and the bisection tree to store the bisection edges.

2.2.1 Unique mid-vertex identifiers: multi-ids

We use *multi-ids* to uniquely identify the new vertices that are created during the bisection process. A multi-id is a sorted list of vertices, $\mathbf{v} = [v_1, \ldots, v_k]$, where $v_1 \leq v_2 \leq \ldots \leq v_k$. Particularly, a vertex id v can be mapped to a multi-id of length one as $[v]$. A simplex that contains multi-ids is denoted as $\sigma = (\mathbf{v}_0, \ldots, \mathbf{v}_n)$. We reinterpret a simplex $\sigma = (v_0, \ldots, v_n)$ using multi-ids as $\sigma = ([v_0], \ldots, [v_n])$.

When creating a new vertex after bisecting and edge, we generate a multi-id for the new vertex. The new multi-id is the combination of the multi-ids of the edge vertices, $\mathbf{v}_0$ and $\mathbf{v}_1$, see Algorithm 2.6. That is, $\mathbf{v}$ is the multi-id of the new vertex after bisecting $e = (\mathbf{v}_0, \mathbf{v}_1)$. The resulting multi-id is created by merging and sorting the the multi-ids of $\mathbf{v}_0$ and $\mathbf{v}_1$. We remark that the ids can appear more than once after generating a new multi-id.

We sort the multi-ids using a lexicographic order. Let $\mathbf{v}_i = [v_{i_1}, \ldots, v_{i_{k_i}}]$ and $\mathbf{v}_j = [v_{j_1}, \ldots, v_{j_{k_j}}]$ be two multi-ids. We say that $\mathbf{v}_i < \mathbf{v}_j$ if there exists r such that $v_{i_l} < v_{j_l}$ for all $l < r$ and $v_{i_l} = v_{j_l}$.

After introducing the notion of multi-ids, we introduce the order of edges for a given simplex. Let $\sigma = ([v_0], [v_1], \ldots, [v_n])$ be a n-simplex, where v_i represents the i-th vertex of σ. We define the list of local edges of σ as $\mathcal{E}$, a sorted list composed of $n(n+1)/2$ edges. We denote as $\mathcal{E}(i)$ the i-th edge of $\mathcal{E}$. For instance, the edges of

Algorithm 2.7 Local edges of a simplex

input: n-`Simplex` σ
output: Sorted list of local edges $\mathcal{E}$

```
1: function getLocalEdges(σ)
2:     ([v_0], [v_1], ..., [v_n]) = σ
3:     E = ∅
4:     for i = 0, ..., n − 1 do
5:         for j = i + 1, ..., n do
6:             e = ([v_i], [v_j])
7:             E = E ∪ e
8:         end for
9:     end for
10:    return E
11: end function
```

the n-simplex σ obtained using Algorithm 2.7 are

$$\begin{array}{ccccccc}
([v_0],[v_1]) & ([v_0],[v_2]) & ([v_0],[v_3]) & & \cdots & & ([v_0],[v_n]) \\
 & ([v_1],[v_2]) & ([v_1],[v_3]) & & \cdots & & ([v_1],[v_n]) \\
 & & ([v_2],[v_3]) & & \cdots & & ([v_1],[v_n]) \\
 & & \ddots & & \vdots & & \vdots \\
 & & & & ([v_{n-2}],[v_{n-1}]) & & ([v_{n-2}],[v_n]) \\
 & & & & & & ([v_{n-1}],[v_n])
\end{array}$$

2.2.2 Non-conformal simplices: hanging vertices

In the recursive refining strategy, see Algorithm 2.4, we need to detect and refine the non-conformal simplices of the mesh. The non-conformal configurations in the mesh are created when refining the adjacent simplices. Thus, the main idea is to loop on the mesh simplices to detect those edges that overlap with an edge bisected from a neighboring element. This bisected edge features a mid-vertex which is seen as a hanging vertex from the edge of the current simplex. The simplices with at least one edge with a hanging vertex define the refinement set to enforce a conformal mesh.

To get the refinement set, Algorithm 2.8 loops over all the simplices of the mesh. For each simplex, if any of its edges has a hanging mid-vertex, we add the simplex in the refinement set. To check if an edge contains a mid-vertex, we use the multi-ids. For a given edge $e = (\mathbf{v}_0, \mathbf{v}_1)$, we obtain the associated multi-id, $\mathbf{v}$, of the mid-vertex using Algorithm 2.6. If the multi-id $\mathbf{v}$ is in the set of mesh vertices, the edge e

Algorithm 2.8 Simplices with hanging vertices of the mesh $\mathcal{T}$.

input: `Mesh` $\mathcal{T}$
output: `SimplicesSet` $\mathcal{M}$

```
1: function getNonConformalSimplices(T)
2:     M = ∅
3:     V = Vertices(T)
4:     for σ ∈ T do
5:         for e ∈ σ do
6:             (v0, v1) = e
7:             v = midVertex(v0, v1)
8:             if v ∈ V then
9:                 M = M ∪ σ
10:            end if
11:        end for
12:    end for
13:    return M
14: end function
```

contains a hanging mid-vertex and therefore, the simplex defines a non-conformal configuration.

2.2.3 Bisection trees and complete vertex trees

During the bisection process, we select a bisection edge e to perform the bisection of σ. The process of selecting the bisection edge is repeated for the two children, the four grandchildren, and so on. We can encode the bisection process by storing those edges in a binary tree that, at level l, has the edges to be bisected of the descendants corresponding to level l. We define as bisection tree of a simplex, denoted as t, the binary tree that in each level contain the edges to bisect of the descendants corresponding to that level. This idea was introduced by Maubach (1995) and is used by Alkämper et al. (2018). Particularly, Alkämper *et al.* show that the bisection tree generated by Maubach (1995), or Traxler (1997), defines a binary tree t of height n for each simplex σ such that t contain all and only the edges of σ. Moreover, Maubach's and Traxler's algorithms lead to a conformal configuration after n uniform refinements.

We define a *balanced bisection tree* of a simplex σ as a bisection tree that has height n, contains all and only the edges of σ, and that after n uniform refinements the generated mesh, composed of 2^n simplices, is conformal. However, there exist

bisection methods that do not generate balanced bisection trees. We define as *balanced bisection method* as a bisection method that for a given simplex σ, after n uniform refinements, the mesh $\mathcal{Q}_n^\sigma$ is conformal and composed of 2^n simplices.

Remark 2.4 (Balanced methods bisect all the edges)**.** Since a balanced bisection tree contains all the edges of a simplex σ, a balanced bisection method bisects all the initial edges after n uniform refinements. This balanced behavior is preferred in local refinement because the resulting edge size is divided by two.

Remark 2.5 (Balanced trees for newest vertex bisection)**.** The bisection trees generated by Maubach (1995) and Traxler (1997) for the tags $d = n$ or $d = n - 1$, are balanced bisection trees, see Lemma 9 and Lemma 11 of Alkämper et al. (2018).

Remark 2.6 (Balanced newest vertex bisection conditions)**.** According to Remark 2.1, Remark 2.2, and Remark 2.5, to obtain a balanced, conformal, finite, stable, and local refinement method using newest vertex bisection, it is sufficient to have a conformal and reflected mesh with all the elements marked as either $d = n$ or $d = n - 1$. Consequently, these are the conditions favored in this thesis because we pursue balanced bisection methods with the advantages of newest vertex bisection.

If we always select as the bisection edge the shortest edge of a simplex σ, the bisection tree t may contain edges that are not defined in σ. In Figure 2.2, we illustrate the bisection tree generated by two different bisection methods when we apply them to an equilateral triangle $\sigma = ([v_0], [v_1], [v_2])$. If we apply Maubach's algorithm with tag $d = 2$ to the triangle σ, we obtain the bisection tree of Figure 2.2(a), that is balanced because has height 2, contain all and only the edges of σ, and leads to a conformal triangular mesh after two uniform refinements, see Figure 2.2(c). If we consider the shortest edge bisection method and the initial bisection edge $([v_0], [v_1])$, the generated bisection tree t, see Figure 2.2(b), has height 2, leads to a conformal triangular mesh after two uniform refinements, see Figure 2.2(d), but is not balanced since t has at level 1 the edges $([v_0], [v_0, v_1])$ and $([v_0, v_1], [v_1])$, that are not defined by the vertices of σ.

Let σ be a simplex and consider that its bisection tree t is balanced. We have seen that if we bisect an edge $([v_i], [v_j])$ the new generated vertex is $[v_i, v_j]$ with $v_i < v_j$. Thus, there is a direct relation between an edge $e_k = ([v_{1,k}], [v_{2,k}])$, where e_k is an edge of level k of t, and the new vertex $[v_{1,k}, v_{2,k}]$ that is generated after bisecting e_k.

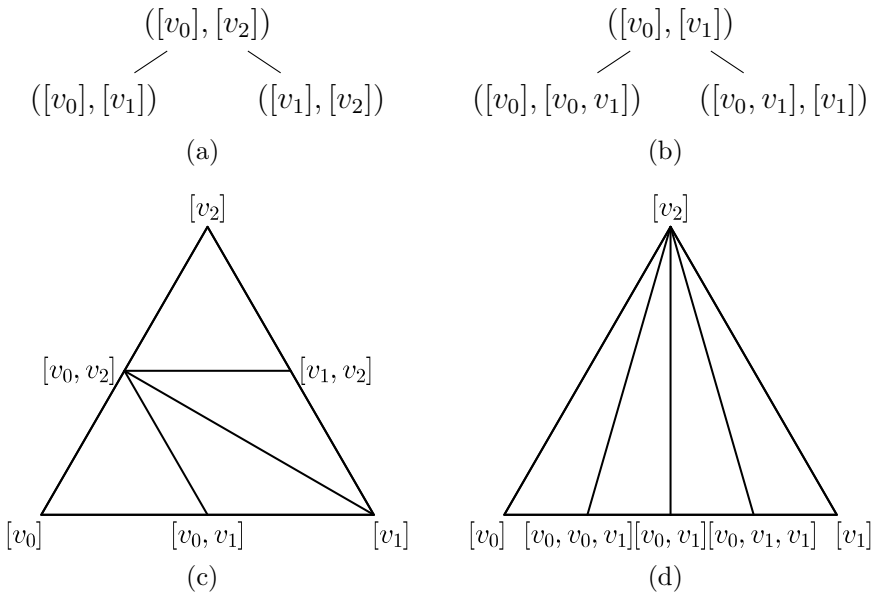


Figure 2.2: Bisection trees of the triangle $\sigma = ([v_0],[v_1],[v_2])$ and generated meshes after two uniform refinements. The bisection tree (a) generated by Maubach's algorithm considering σ as a tagged triangle with tag $d = 2$ generates the triangular mesh (c). The bisection tree (b) generated by shortest edge bisection, being $([v_0],[v_1])$ the first bisection edge, algorithm generates the mesh (d).

We define the *balanced vertex tree*, denoted as $\widehat{t}$, as the binary tree that in each level contain the new vertices that are generated during the bisection process.

Let t be a balanced bisection tree and consider a branch $\{e_0, e_1, \ldots, e_{n-1}\}$ of t, where the sub-index in e_l determines the level l of the edge in the tree. We can identify the branch of edges $\{e_0, e_1, \ldots, e_{n-1}\}$ with the branch of new vertices $\{[v_{1,0}, v_{2,0}], [v_{1,1}, v_{2,1}], \ldots, [v_{1,n-1}, v_{2,n-1}]\}$ of $\widehat{t}$, where the first index of the vertices determines if it is the first or second vertex of the edge, and the second index determines the level of the edge. This branch determines the $(n-1)$-simplex κ

$$\kappa = \{[v_{1,0}, v_{2,0}], [v_{1,1}, v_{2,1}], \ldots, [v_{1,n-1}, v_{2,n-1}]\}.$$

Moreover, κ is an interior face due the fact that its generated only by mid-vertices. After bisecting the last edge e_{n-1} the two generated simplices are

$$\sigma_1 = \{[v_{1,0}, v_{2,0}], \ldots, [v_{1,n-1}, v_{2,n-1}], [v_{1,n-1}]\},$$
$$\sigma_2 = \{[v_{1,0}, v_{2,0}], \ldots, [v_{1,n-1}, v_{2,n-1}], [v_{2,n-1}]\},$$

that share the interior face κ.

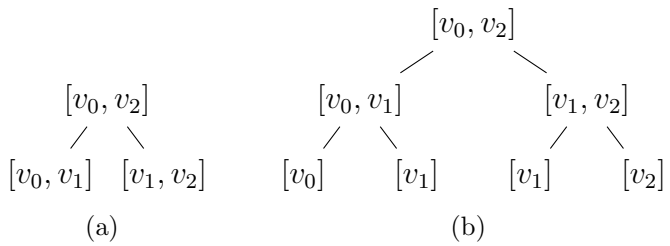


Figure 2.3: Balanced vertex tree and its complete vertex tree. (a) Vertex tree and (b) complete vertex tree associated to Figure 2.2(a).

Consider that we add one more level of vertices to $\widehat{t}$, being this level composed of the vertices of the edges of the level $n-1$. We generate a tree of vertices such that each branch defines a simplex. This tree of vertices is defined as *completed vertex tree*. In particular, since the height of this completed vertex tree is $n+1$, it defines 2^n simplices, that correspond to the generated simplices after n uniform refinements.

Consider again the equilateral triangle and the application of Maubach's algorithm with $d = 2$. We illustrate in Figure 2.3 the tree of vertices and its extension tree, see Figures 2.3(a) and 2.3(b), respectively, associated to the balanced bisection tree of Figure 2.2(a). We see that the nodes of the vertex tree of Figure 2.3(a) are the new vertices created during the bisection process. Moreover, the branches $\{[v_0, v_2], [v_0, v_1]\}$ and $\{[v_0, v_2], [v_1, v_2]\}$ of Figure 2.3(a) are inner faces the triangular mesh of Figure 2.2(c). Finally, the four branches of the extended tree of vertices of Figure 2.3(b), that are $\{[v_0, v_2], [v_0, v_1], [v_0]\}$, $\{[v_0, v_2], [v_0, v_1], [v_1]\}$, $\{[v_0, v_2], [v_1, v_2], [v_1]\}$ and $\{[v_0, v_2], [v_1, v_2], [v_2]\}$, correspond to the four triangles generated after two uniform refinements.

Chapter 3

Marked bisection in n dimensions

The question of whether there is a practical n-dimensional multi-stage bisection method for unstructured conformal meshes is still open. To formally answer this question is the goal of Chapter 6. In this chapter we want to heuristically propose a practical n-dimensional multi-stage bisection on unstructured conformal meshes.

To meet our goal, the main contribution is to propose and implement an n-dimensional three-stage bisection method. By construction, the method starts by marking the sub-simplices of the initial mesh conformingly. Then, independently for each simplex, the first $n-1$ marked bisections accumulate in reverse order the list of newly created mid-edge vertices (first stage). The n-th marked bisection completes the accumulated vertex list to replace simplices of bisection level n with equivalent simplices having a Maubach tag equal to n (second stage). This equivalent tagging allows switching to Maubach's newest vertex bisection (third stage). The pre-process and the three stages work in n dimensions by construction, and hence, the method is n-dimensional. Furthermore, the method heuristically enforces conformity and finiteness while almost meeting the similarity and locality properties. Although we do not formally prove that the conformity and reflectivity conditions are satisfied, we perform the corresponding experiments to check them. We finally apply the implementation of the proposed method to successfully locally refine unstructured conformal meshes.

Algorithm 3.1 Bisection of a marked simplex ρ.

input: `MarkedSimplex` ρ
output: `MarkedSimplex` ρ_1, `MarkedSimplex` ρ_2

```
1:  function bisectSimplex(ρ)
2:      l = level(ρ)
3:      if l < n − 1 then
4:          τ = TreeSimplex(ρ)
5:          τ1, τ2 = bisectStageOne(τ)
6:          ρ1, ρ2 = MarkedSimplex(τ1, τ2)
7:      else if l = n − 1 then
8:          τ = TreeSimplex(ρ)
9:          μ1, μ2 = bisectCastToMaubach(τ)
10:         ρ1, ρ2 = MarkedSimplex(μ1, μ2)
11:     else
12:         μ = MaubachSimplex(ρ)
13:         μ1, μ2 = bisectMaubach(μ)
14:         ρ1, ρ2 = MarkedSimplex(μ1, μ2)
15:     end if
16:     return ρ1, ρ2
17: end function
```

3.1 Problem and outline of our solution

Our goal is to locally refine an n-dimensional conformal unstructured simplicial mesh using a stable bisection method that ensures a conformal mesh in a finite number of steps. The input of our problem is an unstructured conformal simplicial mesh $\mathcal{T}$ and a set of elements $\mathcal{M}$ to be refined. The output is a conformal unstructured marked simplicial mesh $\mathcal{T}_1$ with the simplices in $\mathcal{M}$ bisected.

The kernel of our bisection method, Algorithm 3.1, bisects a single simplex. This algorithm is divided into three stages determined by the descendant level of the simplex to be bisected. The first stage (Lines 3–6) and the second stage (Lines 7–10) promote that we obtain a conformal reflected mesh. In this manner, we ensure a strongly compatible mesh which is a sufficient condition to use Maubach's algorithm. The last stage is Maubach's algorithm (Lines 11–14). Maubach's algorithm can be applied locally ensuring the generation of conformal meshes in a finite number of steps. In addition, it is the only known bisection strategy for n-dimensional simplicial meshes that has been proved to guarantee a finite number of similarity classes, and therefore, we ensure that our bisection strategy is stable.

3.2 Unequivocal edge selection per mesh entity: consistent bisection edge

For all mesh entities shared by different mesh elements, we must ensure that these entities have the same bisection edge on all those elements. To this end, we base this selection on a strict total order of the mesh edges. The main idea is to order the edges from the longest one to the shortest one, and use a tie-breaking rule for the edges with the same length. Specifically, we define the consistent bisection edge of a simplex as the longest edge with the lowest global index.

A shared edge between two simplices may have a different order of vertices, which can induce different results when computing the edge length from different elements. To avoid these discrepancies, we use the concept of *global edges*. A global edge is a pair of indices, $e = (\mathbf{v}_i, \mathbf{v}_j)$, with $\mathbf{v}_i < \mathbf{v}_j$. The length of a global edge is defined as $\|e\| = \|\Pi(\mathbf{v}_j) - \Pi(\mathbf{v}_i)\|$, where $\|\cdot\|$ denotes the Euclidean norm. We compute the length of an edge using the corresponding global edge. Therefore, for different views of the same global edge, we always obtain the same length since we perform the same numerical operations in the same order.

To define a strict total order of edges, when two edges have the same length, we need a tie-breaking rule. To this end, we use a lexicographic order for the global edges in terms of the order of the vertices. We say that the global edge $e_i = (\mathbf{v}_{i_1}, \mathbf{v}_{i_2})$ has lower global index than the global edge $e_j = (\mathbf{v}_{j_1}, \mathbf{v}_{j_2})$ if $\mathbf{v}_{i_1} < \mathbf{v}_{j_1}$, or $\mathbf{v}_{i_1} = \mathbf{v}_{j_1}$ and $\mathbf{v}_{i_2} < \mathbf{v}_{j_2}$. Note that the proposed lexicographic order is strict and total since it is straight-forward to check that is irreflexive, transitive, asymmetric, and connected. We identify each global edge with a unique integer. To this end, we sort all the existing edges of the mesh using the global index criteria. Then, we sort the edges by length using a stable sorting method. This guarantees that edges with the same length maintain the order given by the lowest global index.

The *consistent bisection edge* of a simplex is the edge with the lowest integer assigned in the edge ordering process. Note that the consistent bisection edge of a simplex is unique because we use a strict total order to define it.

Algorithm 3.2 Mark a k-simplex.

input: k-`Simplex` σ
output: `BisectionTree` t

function `stageOneTree`(σ)
  e = `consistentBisectionEdge`(σ)
  if $\dim \sigma = 1$ **then**
    t = `tree`(`node` = e)
  else
    $([v_1], [v_2]) = e$
    κ_1 = `oppositeFace`(σ, $[v_1]$)
    κ_2 = `oppositeFace`(σ, $[v_2]$)
    t_1 = `stageOneTree`(κ_1)
    t_2 = `stageOneTree`(κ_2)
    t = `tree`(`node` = e, `left` = t_1, `right` = t_2)
  end if
  return t
end function

Algorithm 3.3 Mark a conformal simplicial mesh.

input: `ConformalMesh` $\mathcal{T}$
output: `ConformalMarkedMesh` $\mathcal{T}'$

function `markMesh`($\mathcal{T}$)
  $\mathcal{T}' = \emptyset$
  for $\sigma \in \mathcal{T}$ **do**
    t = `stageOneTree`(σ)
    $\bar{\kappa} = ()$
    $l = 0$
    $\rho = (\sigma, \bar{\kappa}, t, l)$
    $\mathcal{T}' = \mathcal{T}' \cup \rho$
  end for
  return $\mathcal{T}'$
end function

3.3 Pre-processing: codimensional marks

We propose a codimensional marking process for a simplex, in which the resulting mark is a tree. The tree is computed by traversing the sub-entities of the simplex in a recursive manner and selecting the consistent bisection edge of each sub-simplex. The resulting *bisection tree* has height n, and the tree nodes of level i correspond to the consistent bisection edges of sub-simplices of co-dimension i (dimension $n - i$).

Next, we detail the codimensional marking process for a single simplex, Algorithm 3.2. Since the codimensional marking process is the first step of the mesh refinement algorithm, the length of the multi-ids of all simplices is one. The input of the function is a simplex $\sigma = ([v_0], \ldots, [v_n])$ and the output is the corresponding bisection tree. First, we obtain the consistent bisection edge, e, of the simplex, see Line 2. If σ is an edge, this corresponds to the base case of the recursion and we return a tree with only the root node. Otherwise, we obtain the opposite faces of the vertices of the bisection edge, see Lines 7–8. Then, we recursively call the marking process algorithm for the faces κ_1 and κ_2, and we obtain the corresponding trees t_1 and t_2, see Lines 9–10. Finally, we build the bisection tree t with the bisection edge as root node and the trees t_1 and t_2 as left and right branches, see Line 11.

Then, we obtain a marked mesh by marking all mesh simplices, see Algorithm 3.3. The input is a conformal simplicial mesh, $\mathcal{T}$, and the output is a conformal marked simplicial mesh, $\mathcal{T}'$. A *tree-simplex* is a 4-tuple $\tau = (\sigma, \bar{\kappa}, t, l)$ where σ is the original simplex, $\bar{\kappa}$ is a list of vertices, t is the bisection tree of the simplex τ, and l is the bisection level. The marked mesh is composed of tree-simplices. We create an empty marked mesh $\mathcal{T}'$. For each simplex of the original mesh $\mathcal{T}$, we create a marked tree-simplex τ, where t is the bisection tree of σ, $\bar{\kappa}$ is an empty list, and the bisection level is 0. Then, we append the tree-simplex τ to the marked mesh $\mathcal{T}'$.

3.4 First stage: tree-simplices

In the first stage, we bisect the simplices using the bisection trees computed with the codimensional marking process. The first stage is used in the first $n-2$ bisection steps and, therefore, the generated simplices have at most descendant level $n-2$. Moreover, during the refinement process we store the new mid-vertices into $\bar{\kappa}$. Thus, in the second stage, we are able to map the generated simplices into a reflected mesh, a mesh type that it is also strongly compatible. When we refine adjacent simplices using their bisection trees, we will obtain a conformal mesh since, by construction, we enforce that a sub-simplex has the same bisection tree independently of the marked simplex it belongs to.

Algorithm 3.4 details the bisection process of a simplex in the first stage. First, we get the bisection edge, e, taking the root of the bisection tree, t, see Line 3. Then, we call the function that bisects a tree-simplex, see Algorithm 3.5. This function

Algorithm 3.4 Bisect a marked tree-simplex.

input: `TreeSimplex` τ
output: `TreeSimplex` τ_1, `TreeSimplex` τ_2

1: **function** `bisectStageOne`(τ)
2: $(\sigma, \bar{\kappa}, t, l) = \tau$
3: $e = \texttt{root}(t)$ ▷ Bisection edge
4: $\sigma_1, \bar{\kappa}_1, \sigma_2, \bar{\kappa}_2 = \texttt{bisectTreeSimplex}(\sigma, \bar{\kappa}, e, l)$
5: $t_1 = \texttt{left}(t)$; $t_2 = \texttt{right}(t)$ ▷ Bisect tree
6: $l_1 = l + 1$; $l_2 = l + 1$ ▷ Bisect level
7: $\tau_1 = (\sigma_1, \bar{\kappa}_1, t_1, l_1)$
8: $\tau_2 = (\sigma_2, \bar{\kappa}_2, t_2, l_2)$
9: **return** τ_1, τ_2
10: **end function**

Algorithm 3.5 Bisect a tree-simplex.

input: `Simplex` σ, l-`List` $\bar{\kappa}$, `Edge` e, `Level` l
output: `Simplex` σ_1, $(l+1)$-`List` $\bar{\kappa}_1$, `Simplex` σ_2, $(l+1)$-`List` $\bar{\kappa}_2$

1: **function** `bisectTreeSimplex`(σ, $\bar{\kappa}$, e, l)
2: $(\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n) = \sigma$
3: $([v_{1,l-1}, v_{2,l-1}], \ldots, [v_{1,0}, v_{2,0}]) = \bar{\kappa}$
4: $([v_{1,l}], [v_{2,l}]) = e$
5: $[v_{1,l}, v_{2,l}] = \texttt{midVertex}([v_{1,l}], [v_{2,l}])$
6: $(i_1, i_2) = \texttt{simplexVertices}(\sigma, e)$
7: $\sigma_1 = (\mathbf{v}_0, \ldots, \mathbf{v}_{i_2-1}, [v_{1,l}, v_{2,l}], \mathbf{v}_{i_2+1}, \ldots, \mathbf{v}_n)$
8: $\sigma_2 = (\mathbf{v}_0, \ldots, \mathbf{v}_{i_1-1}, [v_{1,l}, v_{2,l}], \mathbf{v}_{i_1+1}, \ldots, \mathbf{v}_n)$
9: $\bar{\kappa}_1 = ([v_{1,l}, v_{2,l}], [v_{1,l-1}, v_{2,l-1}] \ldots, [v_{1,0}, v_{2,0}])$
10: $\bar{\kappa}_2 = ([v_{1,l}, v_{2,l}], [v_{1,l-1}, v_{2,l-1}] \ldots, [v_{1,0}, v_{2,0}])$
11: **return** σ_1, $\bar{\kappa}_1$, σ_2, $\bar{\kappa}_2$
12: **end function**

bisects σ into σ_1 and σ_2, and returns two lists of vertices, $\bar{\kappa}_1$ and $\bar{\kappa}_2$, see Line 4. Next, we proceed to bisect the bisection tree t generating two bisection trees t_1 and t_2, see Line 5, that are the left and right branches of t, respectively. Recall that the branches have one level less than t. We do the same with the level l, and we obtain the levels l_1 and l_2 that are defined as $l+1$, see Line 6. After that, we return the tree simplices τ_1 and τ_2 defined in Line 7 and Line 8, respectively.

To bisect a tree-simplex, we apply Algorithm 3.5. The inputs are a simplex, σ, an l-list of vertices, $\bar{\kappa}$, a bisection edge, e, and a descendant level, l. We extract the vertex ids $(\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n)$ of σ, see Line 2. There are l multi-ids of length two and $n + 1 - l$ multi-ids of length one. This is so because at each bisection, we replace a

Algorithm 3.6 Bisect to Maubach

input: `TreeSimplex` τ
output: `MaubachSimplex` μ_1, `MaubachSimplex` μ_2

1: **function** `bisectToMaubach`(τ)
2: $(\sigma, \bar{\sigma}, t, l) = \tau$
3: $e =$ `root`(t)
4: $\sigma_1, \bar{\kappa}_1, \sigma_2, \bar{\kappa}_2 =$ `bisectTreeSimplex`$(\sigma, \bar{\kappa}, e, l)$
5: $\bar{\sigma}_1, \bar{\sigma}_2 =$ `castToMaubach`$(e, \bar{\kappa}_1, \bar{\kappa}_2)$
6: $d_1 = n$; $d_2 = n$
7: $l_1 = l + 1$; $l_2 = l + 1$
8: $\mu_1 = (\bar{\sigma}_1, d_1, l_1)$
9: $\mu_2 = (\bar{\sigma}_2, d_2, l_2)$
10: **return** μ_1, μ_2
11: **end function**

multi-id of length one with a multi-id of length two, the latter multi-id corresponding to the mid-vertex. In Line 3, we extract the vertex ids of the vertices list.

Since the bisection edge belongs to the initial simplex, the length of the multi-ids of the vertices are one. Thus, we write $e = ([v_{1,l}], [v_{2,l}])$, see Line 4, and we create the new mid-vertex $[v_{1,l}, v_{2,l}]$ in Line 5. The new mid-vertex is identified using a multi-id of length two. Then, we obtain the local identifier of the vertices of e inside the simplex σ, see Line 6. After that, we define as children of σ the simplices σ_1 and σ_2. In σ_1 we replace the i_2-th local vertex by the new vertex $[v_{1,l}, v_{2,l}]$, see Line 7. We proceed in the same manner for σ_2 substituting the i_1-th local vertex by the new vertex, see Line 8. Next, we add the new vertex $[v_{1,l}, v_{2,l}]$ at the beginning of the lists $\bar{\kappa}_1$ and $\bar{\kappa}_2$, see Lines 9 and 10, respectively.

Note that $\bar{\kappa}_1$ and $\bar{\kappa}_2$ are the same list on the current bisection step. Nevertheless, in the next bisection step, we may append a different vertex to each list. Moreover, a tree-simplex with descendant level l has a sorted list composed of l vertices. Since a tree-simplex descendant level is at most $n - 1$, the list of vertices contains at most $n - 1$ vertices.

3.5 Second stage: casting to Maubach

We next introduce the second stage of our bisection method for simplices. In this stage, after bisecting a simplex, we reorder the vertices of the bisected simplices to

Algorithm 3.7 Cast to Maubach.

input: `Edge` e, n-`List` $\bar{\kappa}_1$, n-`List` $\bar{\kappa}_2$
output: n-`Simplex` $\bar{\sigma}_1$, n-`Simplex` $\bar{\sigma}_2$
1: **function** `castToMaubach`(e, $\bar{\kappa}_1$, $\bar{\kappa}_2$)
2: $([v_{1,n-1}], [v_{2,n-1}]) = e$
3: $([v_{1,n-1}, v_{2,n-1}], \ldots, [v_{1,0}, v_{2,0}]) = \bar{\kappa}_1$
4: $([v_{1,n-1}, v_{2,n-1}], \ldots, [v_{1,0}, v_{2,0}]) = \bar{\kappa}_2$
5: $\bar{\sigma}_1 = ([v_{1,n-1}], [v_{1,n-1}, v_{2,n-1}], \ldots, [v_{1,0}, v_{2,0}])$
6: $\bar{\sigma}_2 = ([v_{2,n-1}], [v_{1,n-1}, v_{2,n-1}], \ldots, [v_{1,0}, v_{2,0}])$
7: **return** $\bar{\sigma}_1$, $\bar{\sigma}_2$
8: **end function**

obtain a reflected mesh. Thus, in the third stage, we can apply Maubach's algorithm to further refine the mesh.

The second stage is used when the descendant level of a tree-simplex, τ, is $l = n-1$. At this step, $\bar{\kappa}$ is a $(n-1)$-list of vertices that have been accumulated in the previous $(n-1)$ steps of the first stage. Finally, t is a bisection tree that only contains a single vertex. That is, t is a leaf where the consistent bisection edge $e = ([v_{1,n-1}], [v_{2,n-1}])$ is the root.

In the second stage of our proposed bisection method, the input is a tree-simplex $\tau = (\sigma, \bar{\kappa}, t, l)$, and the outputs are two Maubach simplices μ_1 and μ_2. A Maubach simplex is a 3-tuple $\mu = (\bar{\sigma}, d, l)$, where $\bar{\sigma}$ is an equivalent simplex, but it is reordered to properly contribute to a reflected mesh, d is a Maubach integer tag, and l is the descendant level.

First, we obtain the consistent bisection edge, $e = ([v_{1,n-1}], [v_{2,n-1}])$, Line 3. Then, we bisect the simplex σ using the function that bisects tree-simplices, generating two simplices σ_1 and σ_2 and the sorted n-lists of vertices $\bar{\kappa}_1$ and $\bar{\kappa}_2$. After that, we reorder the simplices to be able to apply Maubach's algorithm, see Line 5. After generating the two simplices $\bar{\sigma}_1$ and $\bar{\sigma}_2$ that define a reflected neighbors configuration, we bisect the descendant level l generating two descendant levels $l_1 = n$ and $l_2 = n$, see Line 7. Then, we set the Maubach tags $d_1 = n$ and $d_2 = n$, see Line 6. According to Remark 2.6, we can also use the tag $d = n - 1$. Finally, we create the Maubach simplices $\mu_1 = (\bar{\sigma}_1, d_1, l_1)$ and $\mu_2 = (\bar{\sigma}_2, d_2, l_2)$.

To cast the simplices to obtain a reflected configuration, we apply Algorithm 3.7. The inputs are the bisection edge, and two lists of vertices. The outputs are two reordered simplices in a reflected configuration. This algorithm adds the vertices

Algorithm 3.8 Adapted Maubach's algorithm.

input: `MaubachSimplex` μ
output: `MaubachSimplex` μ_1, `MaubachSimplex` μ_2

1: **function** `bisectMaubach`(μ)
2: $((\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n), d, l) = \mu$
3: $\mathbf{w} = \texttt{midVertex}(\mathbf{v}_0, \mathbf{v}_d)$
4: $\bar{\sigma}_1 = (\mathbf{v}_0, \ldots, \mathbf{v}_{d-1}, \mathbf{w}, \mathbf{v}_{d+1}, \ldots, \mathbf{v}_n)$
5: $\bar{\sigma}_2 = (\mathbf{v}_1, \ldots, \mathbf{v}_d, \mathbf{w}, \mathbf{v}_{d+1}, \ldots, \mathbf{v}_n)$
6: Set $d' = \begin{cases} d-1, & d > 1 \\ n, & d = 1 \end{cases}$
7: $d_1 = d'$; $d_2 = d'$
8: $l_1 = l + 1$; $l_2 = l + 1$
9: $\mu_1 = (\bar{\sigma}_1, d_1, l_1)$
10: $\mu_2 = (\bar{\sigma}_2, d_2, l_2)$
11: **return** μ_1, μ_2
12: **end function**

of the bisection edge $[v_{1,n-1}]$ and $[v_{2,n-1}]$ to $\bar{\kappa}_1$ and $\bar{\kappa}_2$, respectively, generating two sorted $(n+1)$-list. The lists $\bar{\kappa}_1$ and $\bar{\kappa}_2$ contain the same vertices that σ_1 and σ_2, respectively, but in a different order. The order of vertices induced by $\bar{\kappa}_1$ and $\bar{\kappa}_2$ leads to a reflected mesh. Next, we create the simplices $\bar{\sigma}_1$ and $\bar{\sigma}_2$ by casting the $(n+1)$-sorted list of vertices $\bar{\kappa}_1$ and $\bar{\kappa}_2$ into simplices, respectively.

3.6 Third stage: Maubach's bisection

Following, we describe the third stage of our bisection algorithm. In this stage, we use Maubach's algorithm to favor the conformity, finiteness, stability, and locality properties.

We reinterpret Maubach's algorithm using tagged simplices and multi-ids in Algorithm 3.8. The input is a Maubach simplex, $\mu = (\bar{\sigma}, d, l)$, and the outputs are two Maubach simplices, μ_1 and μ_2. First, we generate the new vertex $\mathbf{w}$ as the mid-vertex of $\mathbf{v}_0$ and $\mathbf{v}_d$, see Line 3. That is, the bisection edge is $e = (\mathbf{v}_0, \mathbf{v}_d)$. Then, we bisect $\bar{\sigma}$ and generate the children $\bar{\sigma}_1$ and $\bar{\sigma}_2$, see Lines 4 and 5, respectively. After that, we set the new tag d' for the children simplices, see Line 6. Thus, we define the tags d_1 and d_2 as d', see Line 7. Analogously, we bisect the levels l_1 and l_2, see Line 8. Finally, we create two Maubach simplices μ_1 and μ_2, see Lines 9 and 10, respectively.

In this algorithm, we use the notation $\sigma = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n)$ because any of the

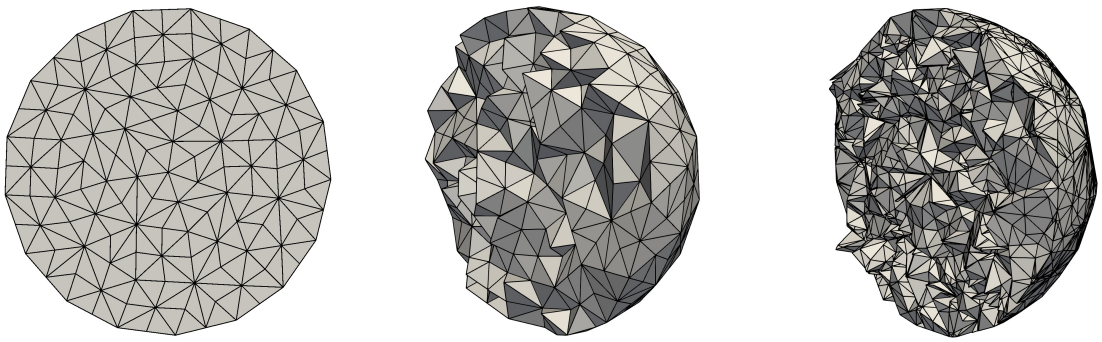
multi-ids can be of length two or higher. This is so since if we call local refine after previous local refinements, our marked mesh might contain Maubach simplices with tags in the range $1 \leq d \leq n-2$. For this tag range, Maubach's algorithm bisects edges of newer generations before bisecting all the original edges of the Maubach simplex (Alkämper et al., 2018). Therefore, we can have multi-ids of length higher than two.

3.7 Examples

We present several examples to illustrate that our proposed algorithm bisects unstructured simplicial meshes, locally adapts conformal meshes, generates a finite number of similarity classes, and leads to lower-bounded quality meshes. For all the examples, we have computed the shape quality of the mesh (Knupp, 2001). We plot the minimum and maximum shape quality of the mesh in each refinement step to illustrate that the minimum quality is lower bounded and cycles.

To validate the results, we need the capabilities to check the conformity and the reflectivity of some of the meshes generated during the refinement process. To check the conformity, we check that all the interior faces of $\mathcal{T}_k$ are shared only by two simplices and that all the boundary faces of mesh $\mathcal{T}_k$ are descendants of the original boundary faces. See more details in C.1. To check the reflectivity of $\mathcal{T}_k$, we check that all the neighboring simplices are reflected neighbors through the shared face. See more details in C.2.

All the results have been obtained on a MacBook Pro with one dual-core Intel Core i5 CPU, with a clock frequency of 2.7GHz, and a total memory of 16GBytes. As a proof of concept, a mesh refiner has been fully developed in Julia 1.4. The Julia prototype code is sequential (one execution thread), corresponding to the implementation of the method presented in this chapter. All the unstructured initial meshes are generated with the `distmesh` algorithm (Persson and Strang, 2004), and all the structured initial meshes are generated with the Coxeter-Freudenthal-Kuhn algorithm (Coxeter, 1934; Freudenthal, 1942; Kuhn, 1960). We recall that after each refinement to conformity, we perform a renumber of the vertices of the mesh $\mathcal{T}_k$ in order to have only multi-ids of length one, see C.3.



(a) (b) (c)

Figure 3.1: Conformal reflected n-dimensional meshes after the first two bisection stages. (a) Conformal reflected triangular mesh $\mathcal{T}_2^2$ after two uniform refinements. (b) Conformal reflected tetrahedral mesh $\mathcal{T}_3^3$ after three uniform refinements. (c) Volume slice with the hyperplane $t = 0$ of the conformal reflected pentatopic mesh $\mathcal{T}_4^4$ after four uniform refinements.

3.7.1 First two stages on unstructured meshes: conformity and reflectivity

The goal of this example is to propose a methodology to check the correctness of the algorithm implementation. Thus, let $\mathcal{T}_0^n$ be a conformal unstructured n-simplicial mesh and consider n uniform refinements using our proposed bisection algorithm. That is, we apply our bisection method until the second stage is executed. At this point of the algorithm, we have reordered the vertices of the simplices and we should obtain a conformal and reflected mesh. We check if the generated n-dimensional mesh $\mathcal{T}_n^n$ is conformal and reflected, conditions that are sufficient to apply Maubach's algorithm. The proposed algorithms to check the conformity and reflectivity of $\mathcal{T}_n^n$ are depicted in C.1 and C.2, respectively.

To verify our proposed algorithm, we present an unstructured example in different dimensions that allows us to check that the obtained results are the expected ones. Let $\bar{B}^n$ be the n-dimensional closed ball of radius 1 centered at the origin, defined as the points such that $\|\mathbf{x}\| \leq 1$. We approximate the domains $\bar{B}^2$, $\bar{B}^3$, $\bar{B}^4$, and $\bar{B}^5$ with the simplicial meshes $\mathcal{T}_0^2$, $\mathcal{T}_0^3$, $\mathcal{T}_0^4$, and $\mathcal{T}_0^5$, respectively. To obtain equivalent mesh resolution, we set the edge length to 0.3 in all the cases.

After generating $\mathcal{T}_0^n$ for each dimension, we apply n uniform refinements using our bisection algorithm, and we obtain the meshes $\mathcal{T}_n^n$ for $n = 2, 3, 4$, and 5. The

meshes $\mathcal{T}_n^n$ have $2^n N^{n_e}$ simplices, where N^{n_e} is the number of simplices of the initial mesh $\mathcal{T}_0^n$. In Figures 3.1(a) and 3.1(b), we show the meshes obtained for the two- and three- dimensional cases, respectively. Moreover, in Figure 3.1(c), we show a 3-dimensional slice of the 4-dimensional mesh.

After the second stage, all the meshes are conformal and reflected, and thus they are strongly compatible. For this reason, we can apply the third stage, which is Maubach's algorithm, to further refine the meshes.

3.7.2 Uniform bisection: minimum quality is lower bounded and cycles

Following, we show that the minimum quality is lower bounded and cycles. To this end, we uniformly refine a single simplex several times. To illustrate the quality cycles, we perform a series of refinements to ensure that the method completes two additional cycles of Maubach's method.

Figure 3.2 plots the evolution of the minimum (blue line) and maximum (red line) qualities during the uniform refinement process. Each column of Figure 3.2 corresponds to a dimensional case, starting from 2D and ending in 4D. Analogously, the first and second rows of Figure 3.2 correspond to an equilateral and an irregular simplex, respectively. Note that in all cases, the minimum quality is lower-bounded, and the maximum and minimum qualities cycle with a period of n steps.

For the 2-dimensional case, we illustrate in Figures 3.2(a) and 3.2(b) the evolution of the minimum and maximum qualities of the meshes obtained by uniformly bisecting an equilateral triangle and an irregular triangle, respectively. For this case, we perform six uniform refinements. Since our method marks a triangle as a tagged triangle with $d = 1$ or $d = 2$, it is analogous to Maubach's algorithm and, with two uniform refinements, all the similarity classes of a triangle are generated. At the second iteration, both the minimum and the maximum qualities start to cycle with a period of two steps.

For the 3-dimensional case, we show in Figures 3.2(c) and 3.2(d) the evolution of the minimum and maximum qualities of the meshes obtained by uniformly bisecting an equilateral tetrahedron and an irregular tetrahedron, respectively. We perform twelve uniform refinements to generate all the similarity classes. For the equilateral tetrahedron, the marking process generates a bisection tree that is equivalent to a tagged tetrahedron with tag $d = 2$ or, equivalently, a planar tetrahedron P_u, as

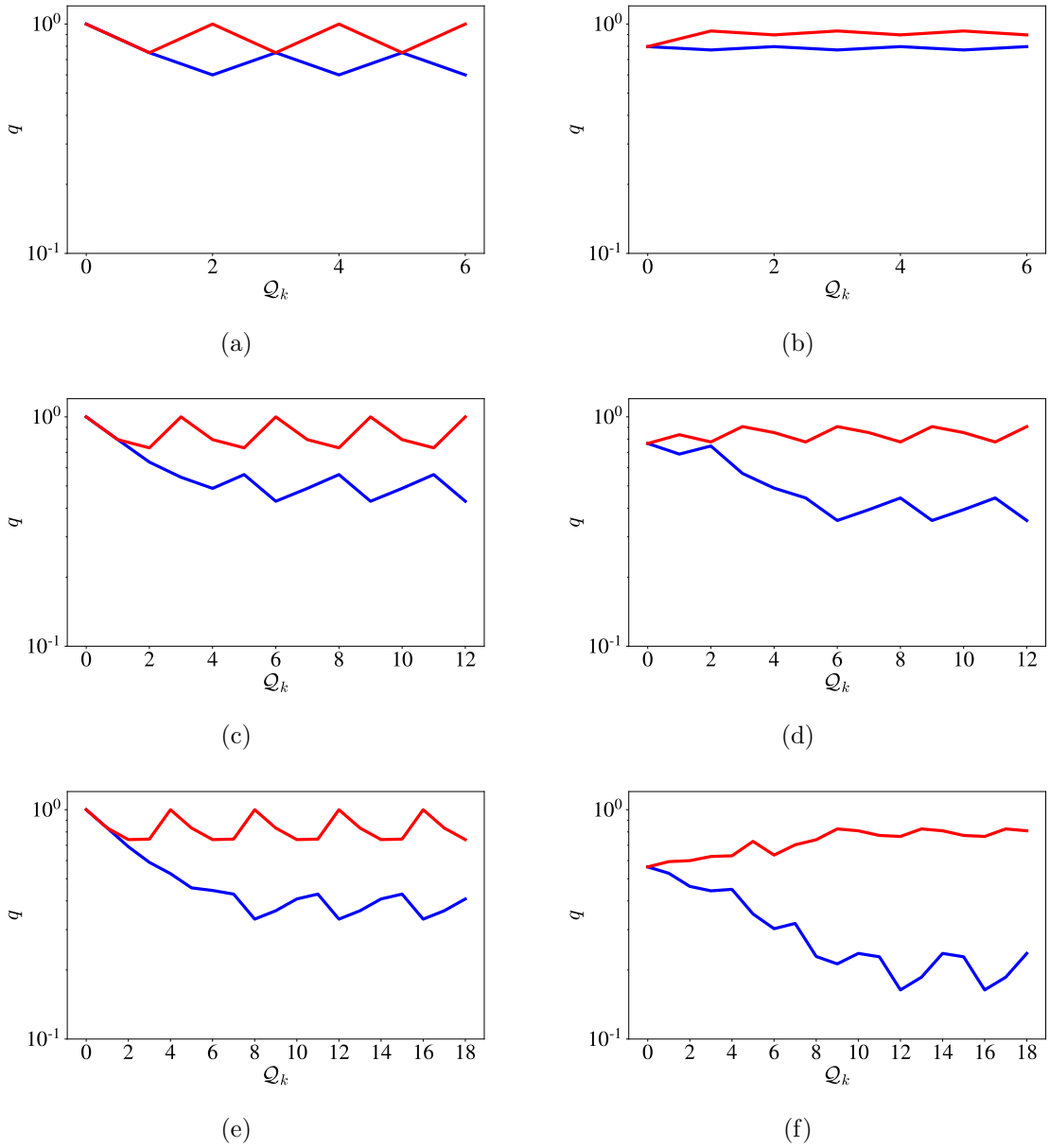


Figure 3.2: Minimum (blue) and maximum (red) quality cycles for uniform refinements. In columns, initial simplex: (a), (c) and (e) equilateral, (b), (d) and (f) irregular simplex. In rows, simplex dimension: (a) and (b) triangles; (c) and (d) tetrahedra; (e) and (f) pentatopes.

denoted in Arnold et al. (2000). At iteration six, the process achieves the minimum quality, and both the minimum and maximum qualities cycle with a period of three steps, see Figure 3.2(c). We obtain similar results for the irregular tetrahedron, see Figure 3.2(d). Specifically, the minimum quality is achieved at iteration six, and both

the minimum and maximum qualities cycle with a period of three steps.

For the 4-dimensional case, we show in Figures 3.2(e) and 3.2(f) the evolution of the minimum and maximum qualities of the meshes obtained by uniformly bisecting an equilateral pentatope and an irregular pentatope, respectively. We performed eighteen uniform refinements. In the case of the equilateral pentatope, the minimum quality is achieved at iteration eight, and both the minimum and maximum quality cycle with a period of four steps, see Figure 3.2(e). When bisecting the irregular pentatope, the minimum quality is obtained at iteration twelve, and both the minimum and maximum qualities cycle with a period of four steps, see Figure 3.2(f).

In all the cases we perform enough uniform bisection steps to generate all the similarity classes and show some full refinement cycles of length n. Moreover, we reach the minimum and maximum mesh quality in a finite number of steps, qualities that are repeated every n steps. This illustrates that the mesh quality does not degenerate with successive refinement and thus, the method is stable.

3.7.3 Local refine of a 4D structured mesh: equivalency to Maubach's method

The main goal of this example is to illustrate that our algorithm is equivalent to newest vertex bisection when we refine a structured mesh. To this end, we recreate the first example from Maubach (1995) and Arnold et al. (2000) but we extend it to four dimensions. Let $[0,1]^4$ be the unit hypercube and consider its subdivision into 16 sub-hypercubes. We subdivide into 24 pentatopes each sub-hypercube using Coxeter-Freudenthal-Kuhn algorithm (Coxeter, 1934; Freudenthal, 1942; Kuhn, 1960), generating a 4D pentatopic mesh, $\mathcal{T}_0$, composed of 384 pentatopes and 81 vertices.

Let

$$H = \left\{ \left(x - \frac{1}{2}\right)^2 + \left(y - \frac{1}{2}\right)^2 + \left(z - \frac{1}{2}\right)^2 + \left(t - \frac{1}{2}\right)^2 = \frac{1}{16},\, x \geq \frac{1}{2} \right\}$$

be an hemisphere of a hypersphere of radius $1/4$, centered at $(1/2, 1/2, 1/2, 1/2)$ that is embedded in the cube $[0,1]^4$. We want to adapt the pentatopic mesh $\mathcal{T}_0$ to the hemisphere H, thus, we choose as refinement set $\mathcal{M}_k$ the pentatopes of $\mathcal{T}_k$ that intersect with the hemisphere H. That is, $\mathcal{M}_k = \{\sigma \in \mathcal{T}_{k-1} \,|\, \sigma \cap H \neq \emptyset\}$. After 22 iterations of the proposed local refinement process, the mesh $\mathcal{T}_{22}$ is composed

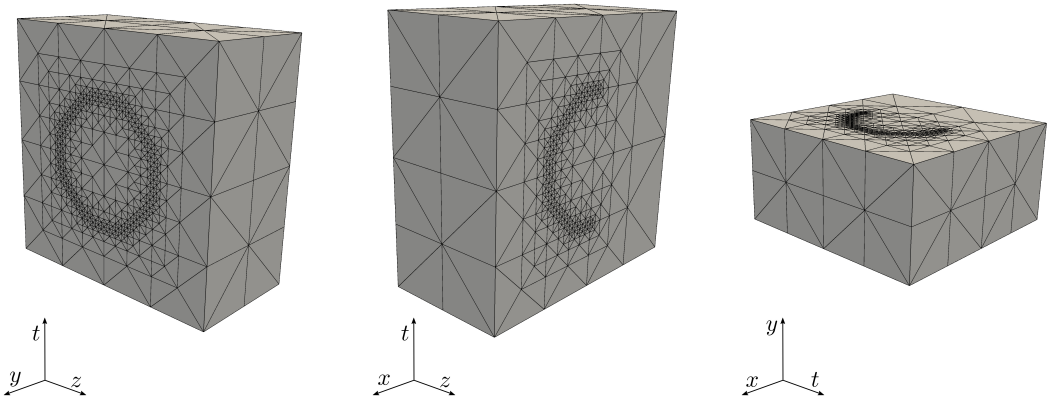


Figure 3.3: Volume slices of 4-dimensional mesh $\mathcal{T}_{22}$ at different planes: (a) $x = 1/2$; (b) $y = 1/2$; and (c) $z = 1/2$.

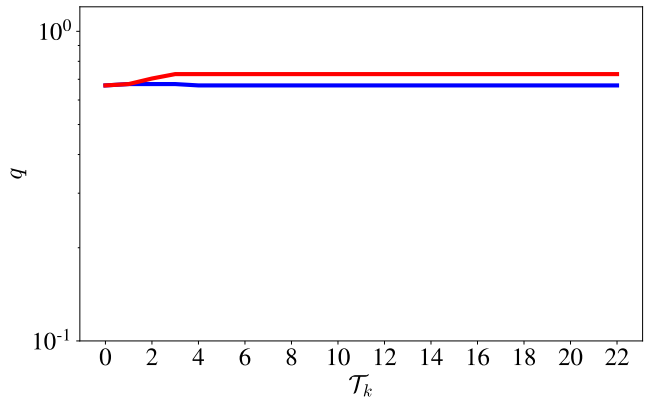
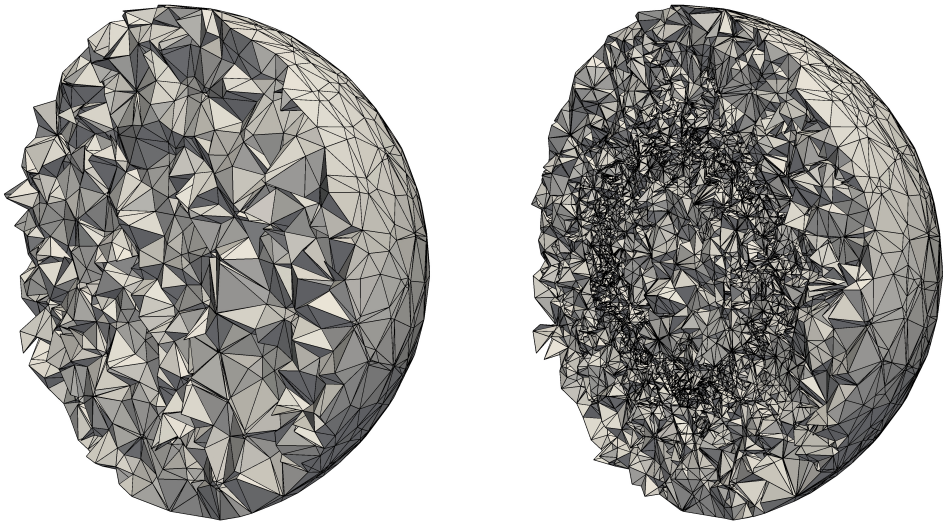


Figure 3.4: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iterations.

of 10093008 pentatopes and 557664 vertices. We make three different slices of the final mesh, $\mathcal{T}_{22}$, with the hyperplanes $x = 1/2$, $y = 1/2$ and $z = 1/2$, see Figures 3.3(a), 3.3(b) and 3.3(c), respectively. We can see how the mesh has been refined locally around the hemisphere. That is, the mesh contains small elements near the hemisphere, and large elements far from the hemisphere.

Figure 3.4 shows the evolution of the maximum and minimum quality of the mesh during the local refinement process. We see that the maximum quality remains constant during the refinement process. The minimum quality of the mesh decreases until



(a) (b)

Figure 3.5: Slice of the 4-simplicial mesh of a hypersphere with the hyperplane $t = 0$: (a) initial mesh; and (b) locally adapted mesh $\mathcal{T}_5$.

iteration 4, and then it stabilizes since, in posterior local refinements, the minimum mesh quality is achieved.

When we apply our codimensional marking process to a Coxeter-Freudenthal-Kuhn mesh, all the initial pentatopes have a bisection tree equivalent to a Maubach pentatope with tag $d = 4$.

3.7.4 Local refine of a 4D unstructured mesh

We show that our bisection method can be applied to locally refine unstructured simplicial meshes. In particular, to 4D unstructured pentatopic meshes. To this end, we generate an unstructured 4D mesh of a hypersphere of radius 1 and centered in the origin. Then, we successively refine those elements that intersect a hypersphere of radius 1/2 and centered in the origin. The initial mesh has an edge length of 0.15 and is composed of 198740 pentatopes and 10361 vertices. Figure 3.5(a) shows a slice of $\mathcal{T}_0$ with the hyperplane $t = 0.0$.

After 5 iterations of the refinement process, the obtained mesh $\mathcal{T}_5$ is composed of 12101892 pentatopes and 614409 vertices. We slice $\mathcal{T}_5$ with the hyperplane $t = 0.0$ to obtain the 3D tetrahedral representation depicted in Figure 3.5(b). We can see how the mesh is refined capturing the inner hypersphere. The obtained results

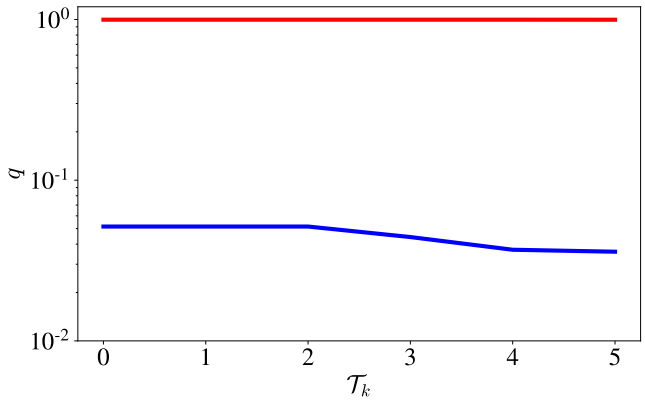


Figure 3.6: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iterations.

illustrate that the proposed bisection algorithm can refine unstructured simplicial meshes locally while preserving conformity.

Figure 3.6 shows the evolution of the shape quality during the refinement process. We see that the maximum quality remains constant during the refinement process. The minimum quality of the mesh decreases but does not achieve its minimum since we need to perform more local refinements.

3.7.5 Local refinement in 4D space-time: time evolution of a 3D potential

We proceed to show that we can locally refine a mesh to capture 3-dimensional manifolds that lay in 4D space-time, manifolds that result from the time evolution of an unsteady 3D potential. We show the evolution of the gravitational potential defined by two mass particles that move along the z-axis. Let

$$V(\mathbf{x}, t) = -G\left(\frac{m_1}{\|\mathbf{x} - \mathbf{p}_1(t)\|} + \frac{m_2}{\|\mathbf{x} - \mathbf{p}_2(t)\|}\right)$$

$$\mathbf{p}_1(t) = \mathbf{p}_1 + (0, 0, vt),\ t \in [0, 1]$$
$$\mathbf{p}_2(t) = \mathbf{p}_2 - (0, 0, vt),\ t \in [0, 1]$$

the equation that defines the gravitational potential. For a given iso-value V_0, $V(\mathbf{x}, t) = V_0$ defines a 3D embedded manifold in 4D space. Let H be the hypercylinder with

spherical basis defined by the equations

$$\left(x-\frac{1}{2}\right)^2+\left(y-\frac{1}{2}\right)^2+\left(z-\frac{1}{2}\right)^2=1$$
$$-0.1\leq t\leq 1.1.$$

In this example, we choose the iso-value $V_0=-10$ and the parameters $G=1$, $m_1=1$, $m_2=1$, $\mathbf{p}_1=(1/2,1/2,1/8)$, $\mathbf{p}_2=(1/2,1/2,7/8)$ and $v=3/8$.

We generate an adapted pentatopic mesh by locally refining an initial mesh around the manifold. We generate the initial mesh $\mathcal{T}_0$ composed of 7345 pentatopes and 576 vertices. We generate the set of pentatopes that intersect H, $F_k=\{\sigma\in\mathcal{T}_{k-1}\,|\,\sigma\cap H\neq\emptyset\}$. Then, for each pentatope in F_k we compute the curvature of $V(\mathbf{x},t)$ at each simplex using the formula

$$e_\sigma=\sum_{i=0}^{4}\left|h_i^T\nabla^2V(\mathbf{x}_i,t_i)h_i\right|,$$

where $\nabla^2V(\mathbf{x}_i,t_i)$ is the Hessian matrix of the potential $V(\mathbf{x},t)$ evaluated at the vertices $(\mathbf{x}_i,t_i)$ of σ, and $h_i=(\mathbf{x}_i,t_i)-c_M$, where c_M is the center of mass of σ. After that, we choose as refinement set $\mathcal{M}_k$ the 10% of the pentatopes of F_k with more curvature. The idea is to adapt the pentatopic mesh not only to the elements that intersect the iso-surface but also to the areas of the iso-surface with more curvature.

After 18 iterations of the local refinement process, the generated mesh $\mathcal{T}_{18}$ has 12115582 pentatopes and 619571 vertices. To visualize the obtained mesh, we sliced it with a hyperplane to obtain a 3D tetrahedral representation. Figures 3.7(a), 3.7(c), and 3.7(e) show a slice of the pentatopic mesh with the hyperplanes $t=0$, $t=0.5$, and $t=1$, respectively. The mesh has been locally refined around the iso-surface and therefore, we have smaller elements near the iso-surface and large elements far from the iso-surface. Figures 3.7(b), 3.7(d), and 3.7(f) show the iso-surface that is extracted from the mesh. These slices show that the iso-surface starts as two different connected components and then it merges as one connected component. Figure 3.8(a) shows a slice of the pentatopic mesh with the hyperplane $x=0.5$, generating the space-time mesh (z,y,t). We can see how the mesh captures the time evolution of the iso-surface defined by $V(\mathbf{x},t)$. Figure 3.8(b) shows the iso-surface that is extracted from the space-time mesh.

Figure 3.9 shows the evolution of the maximum and minimum quality of the mesh during the local refinement process. We see that the maximum quality remains

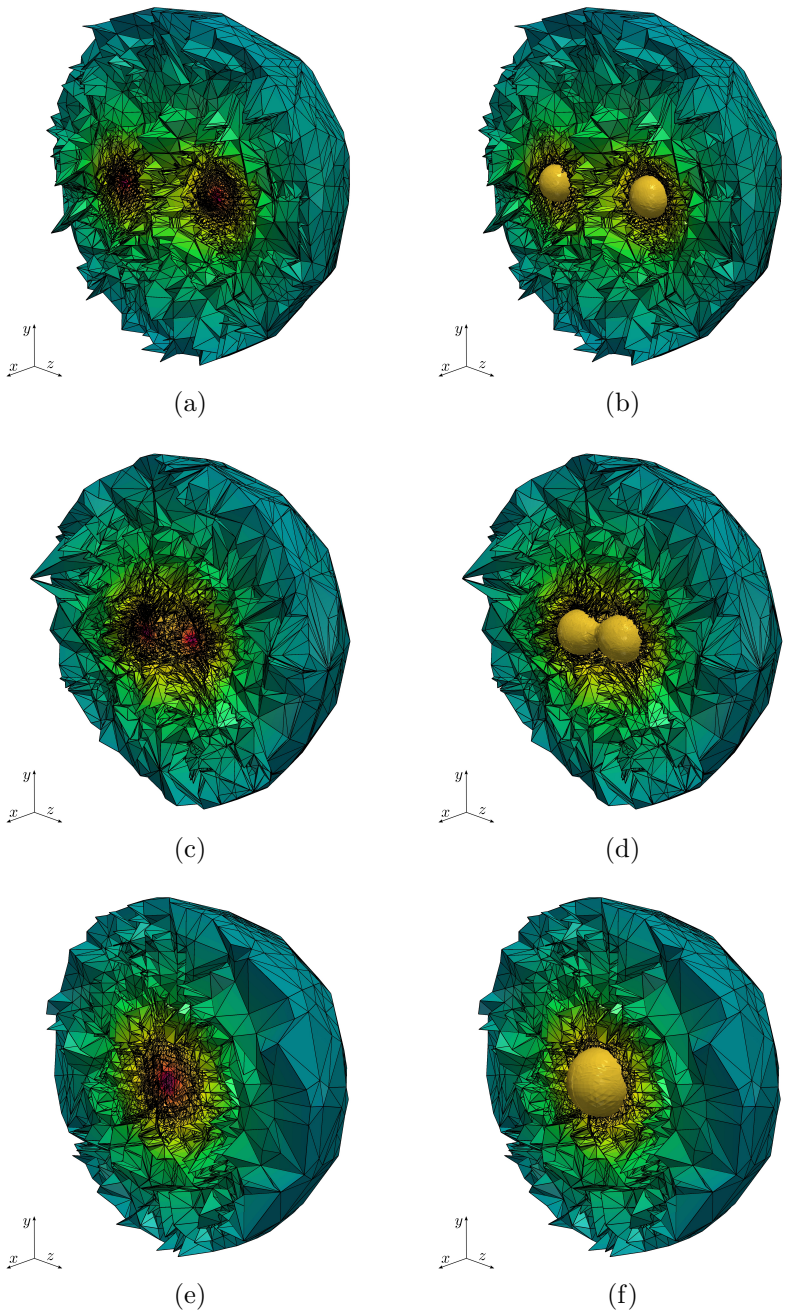
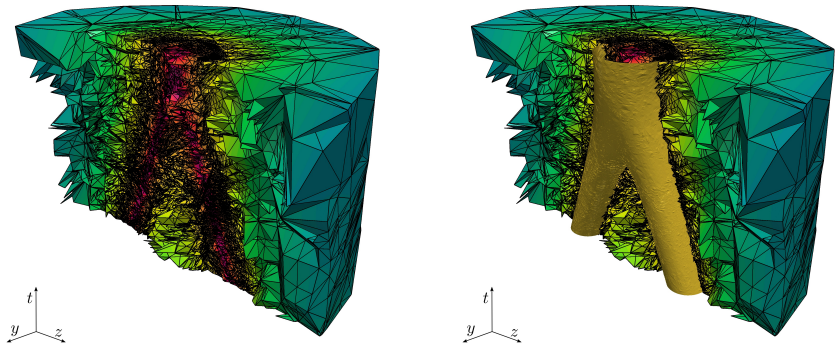


Figure 3.7: Volume slices of $\mathcal{T}_{18}$ at different time instants colored by potential value. In columns, slice of the mesh (a,c,e) without and (b,d,f) with the iso-potential manifold. In rows, slices with: (a,b) $t = 0.0$; (c,d) $t = 0.5$; and (e,f) $t = 1.0$.

constant during the refinement process. The minimum quality of the mesh decreases until iteration 16, and then it stabilizes since, in posterior local refinements, the



(a) (b)

Figure 3.8: Volume slice with the hyperplane $x = 1/2$. In Figures (a) and (b) we obtain the 3D space-time mesh (z, y, t), where we can see the time evolution of the iso-surface defined by the gravitational potential. We can see how the mesh is adapted to capture the movement of the two particles.

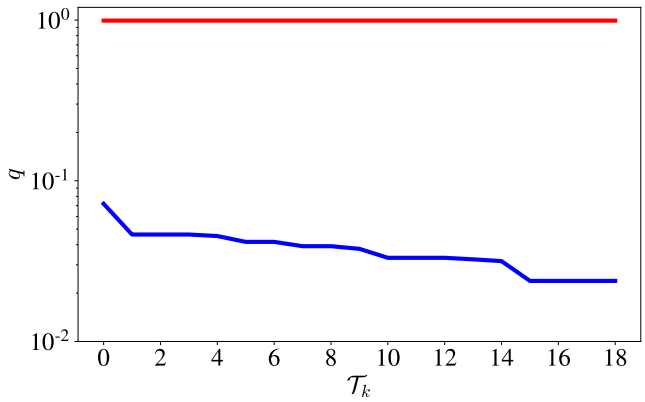


Figure 3.9: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iteration.

minimum mesh quality is achieved.

3.8 Concluding remarks

To heuristically enforce a practical bisection method, we have considered two key ingredients. First, independently for each simplex, the marked bisection is active only in the first n bisection steps. Therefore, it is responsible for a slight initial increase in the number of similarity classes and the cost of completing the conformal closure. Second, the marking process and the two stages heuristically enforce the

condition that after successive n uniform marked bisections, we obtain a conformal and reflected mesh. According to Stevenson (2008), these are sufficient conditions to switch to newest vertex bisection.

In conclusion, we have proposed and implemented the first practical n-dimensional multi-stage bisection for unstructured conformal meshes. In future work, we will prove that the proposed method features the properties of a practical bisection method.

In perspective, guaranteeing unstructured conformal refinement would enable adaptive applications on n-dimensional complex geometry. In this case, the complexity of the geometry would be handled by the flexibility of unstructured conformal meshes. Furthermore, on these meshes, the refinement would almost fulfill the advantages of pure newest vertex bisection for n-simplicial adaption.

Chapter 4

Marked bisection in three dimensions with optimal similarity bound

For three-dimensional unstructured conformal meshes, there are bisection methods with sub-optimal similarity bound (Bänsch, 1991; Kossaczký, 1994; Liu and Joe, 1994, 1995; Arnold et al., 2000). All these methods lead to analogous locally refined conformal meshes. Nevertheless, Arnold et al. (2000) establish a key connection between Maubach (1995) newest vertex bisection and marked bisection of Arnold et al. (2000). Marked bisection leads at most to 72 similarity classes. This bound is two times the number of similarity classes of newest vertex bisection.

To meet this sub-optimal bound, marked bisection of Arnold et al. (2000) features one pre-process stage and two bisection stages. In the pre-processing, for all the faces of the initial mesh, the method conformingly marks the bisection edges. These edge marks determine a finite set of marked tetrahedron types. For each type, there is a specific bisection that leads to two children tetrahedra of the next type. The first stage ensures that different types of tetrahedra are all bisected to the *planar* type. This type, independently for each tetrahedron, is the beginning of the next bisection stage. In this second stage, successive marked bisection cycles every three bisection steps through a subset of the marked types (*unflagged planar*, *flagged planar*, and *adjacent*). This cyclic stage is equivalent to Maubach (1995) newest vertex bisection

under specific conditions stated by Arnold et al. (2000).

The previous overview allows reasoning about the number of similarity classes. On the one hand, the number potentially doubles the bound for the newest vertex bisection due to the initial bisection stage. On the other hand, the cyclic stage guarantees that the rest of the generated similarity classes correspond to those determined by the newest vertex bisection. Accordingly to Arnold et al. (2000), for some conformingly-marked meshes, marked bisection behaves as the newest vertex bisection. Specifically, there are no more than 36 similarity classes if the conformingly-marked mesh is composed only of unflagged planar or adjacent tetrahedra.

The question of whether there is a method to conformingly mark as unflagged planar or as adjacent all the tetrahedra of an arbitrary three-dimensional unstructured conformal mesh is still open, see reference Arnold et al. (2000). A constructive answer is of significant interest. It would lead to the first marked bisection featuring an optimal similarity bound for adaption in complex geometry. The main goal of this chapter is to answer this question and implement the obtained method.

To meet the goal, our main contribution is to propose a new marking procedure for three-dimensional unstructured conformal meshes. For these meshes, we guarantee that all the tetrahedra become conformingly marked as unflagged planar. To this end, we consider three key ingredients. First, we propose a specific ordering of the global mesh edges. Second, relying on this edge ordering, we deduce that all the mesh tetrahedra become marked as unflagged planar. Third, we guarantee conformingly-marked meshes by checking that we fulfill the sufficient conditions for tetrahedral meshes stated by Arnold et al. (2000). To illustrate the application, we implement the refine to conformity marked bisection of Arnold et al. (2000) but equipped with our planar marking method. We use the implementation to locally refine three-dimensional unstructured conformal meshes and check the minimum mesh quality.

4.1 Preliminaries and problem

We proceed to introduce the necessary notation and concepts. Specifically, we introduce the preliminaries related to conformal simplicial meshes and marked bisection. Finally, we state the problem of conformingly marking unstructured simplicial meshes for bisection.

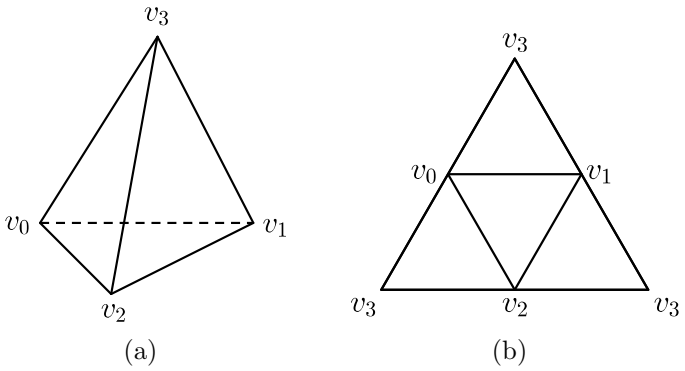


Figure 4.1: Representations of a tetrahedron composed of the vertices v_1, v_2, v_3, and v_4: (a) volumetric; and (b) planar.

4.1.1 Preliminaries and definitions

In our application, we are interested in tetrahedra, three-dimensional simplices. Herein, as Arnold et al. (2000), we represent volumetric tetrahedra composed of the vertices v_1, v_2, v_3, and v_4, see Figure 4.1(a), in the plane by cutting and unfolding the corresponding triangular faces, see Figure 4.1(b).

A tetrahedron has three types of entities: triangles, edges, and vertices, which are sub-simplices composed of 3, 2, and 1 vertices of τ, respectively. We denote the faces, edges and vertices with the letters κ, e and v, respectively. We define the list of local edges of a tetrahedron $\tau = (v_0, v_1, v_2, v_3)$ as the following sorted list of edges

$$(v_0, v_1), (v_0, v_2), (v_0, v_3), (v_1, v_2), (v_1, v_3), (v_2, v_3).$$

We associate each triangular face of a tetrahedron τ with the opposite face to a vertex of τ. As an example, for the tetrahedron $\tau = (v_0, v_1, v_2, v_3)$, the opposite face to the vertex v_0 is the triangular face $\kappa_0 = (v_1, v_2, v_3)$. We say that two tetrahedra τ_1 and τ_2 are *neighbors* if they share a common triangular face.

4.1.2 Marked bisection for tetrahedra

Arnold et al. (2000) presented a marked bisection algorithm for unstructured conformal tetrahedral meshes that ensure locally refined conformal meshes and quality stability. Following, we present the terminology and results required to overview their marked bisection algorithm.

The *refinement edge* e_τ is the edge of τ to be bisected. Since an edge is shared by two triangular faces of the tetrahedron, the triangular faces that contain e_τ are

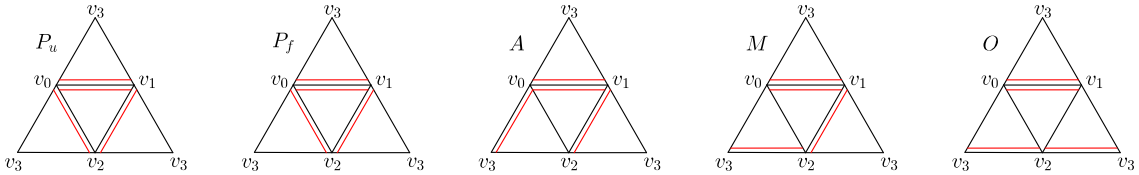


Figure 4.2: The five different type of marked tetrahedra of Arnold's cycle: (a) unflagged planar tetrahedron, (b) flagged planar tetrahedron, (c) adjacent tetrahedron, (d) mixed tetrahedron, and (e) opposite tetrahedron.

the *refinement faces* of τ. The remaining two triangular faces are defined as *non-refinement faces*. For those faces, one edge, referred as *marked edge*, is assigned. We recall that each triangular face κ_i has a refinement edge e_{κ_i}. Particularly, the refinement edge e_τ is the same as the e_{κ_i} of the refinement faces.

Since the non-refinement edges are adjacent or either opposite to the refinement edge, we can classify the marked tetrahedra into four types, see Figure 4.2: adjacent A, planar P, mixed M, and opposite O.

- Planar, P: the refinement edge and the marked edges are coplanar. A planar tetrahedron is further classified as type P_u or type P_f, according to a boolean flag, see Figures 4.2(a), and 4.2(b), respectively.

- Adjacent, A: the marked edges are adjacent to the refinement edge but are not coplanar, see Figures 4.2(c).

- Mixed, M: one marked edge is adjacent to the refinement edge, and the other is opposite, see Figures 4.2(d).

- Opposite, O: both marked edges are opposite to the refinement edge, see Figures 4.2(e).

These tetrahedron types are the nodes of the directed graph determining the marked bisection sequence, see Figure 4.3.

Now, we can introduce the definition of *marked tetrahedron*, which is a modification of the one detailed by Arnold et al. (2000). Herein, a marked tetrahedron is the 5-tuple

$$\rho = (\tau, e_\tau, e_{\kappa_1}, e_{\kappa_2}, t),$$

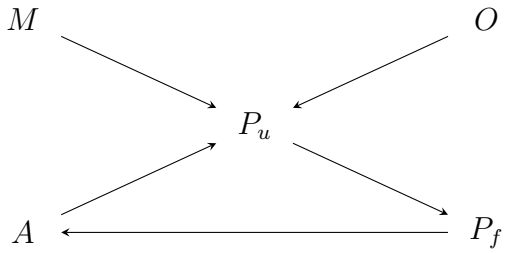


Figure 4.3: Directed graph of tetrahedron types for marked bisection.

where τ is a tetrahedron, e_τ is the refinement edge, e_{κ_1} and e_{κ_2} are the marked edges of the non-refinement faces, and t is the tetrahedron type.

A mesh is *marked* if all its tetrahedra are marked. A marked conformal mesh is *conformingly-marked* if each triangular face has a unique marked edge. That is, a triangular face shared by two tetrahedra has the same marked edge from both sides. Accordingly, shared triangular faces are bisected in the same manner from different tetrahedra.

Remark 4.1 (Conditions to conformingly mark)**.** To guarantee that a conformal mesh is conformingly-marked, Arnold et al. (2000) state that it is sufficient to combine a strict total order of the mesh edges with their marking process for tetrahedra. For instance, the mesh edges can be sorted according to their length using a tie-breaking rule when the lengths are equal.

The marked bisection method, Algorithm 2.2, starts by marking the initial unstructured conformal mesh and then applies a local refinement procedure to a subset of tetrahedra of the marked mesh. The marking pre-process is devised to ensure a conformingly-marked mesh. Using this marked mesh, the local refinement procedure, Algorithm 2.3, first refines a set of tetrahedra and then calls a recursive refine-to-conformity strategy. This strategy, Algorithm 2.4, terminates when successive bisection leads to a conformal mesh. Both algorithms use marked bisection to refine a set of elements, see Algorithm 2.5.

Remark 4.2 (Optimal similarity bound)**.** If the conformingly-marked mesh is composed only of unflagged planar or adjacent tetrahedra, marked bisection does not generate more than 36 similarity classes, see reference Arnold et al. (2000).

4.1.3 Problem

Our problem is to conformingly mark an unstructured conformal tetrahedral mesh $\mathcal{T}_1$ exclusively with tetrahedra of type P_u. Thus, when applying successive marked bisection, starting on the resulting marked $\mathcal{T}_1$, we can guarantee an optimal number of similarity classes, see Remark 4.2. Specifically, starting on the unflagged planar mesh $\mathcal{T}_1$, if we locally refine a set of elements $\mathcal{M}$, we obtain a new conformal unstructured marked tetrahedral mesh $\mathcal{T}_2$ with the corresponding elements bisected. The marked mesh $\mathcal{T}_2$ is suitable for a posterior local refinement. Furthermore, any successive local refinement process has the minimum element quality bounded.

4.2 Solution: conformingly marking as planar

Following, we detail our solution to conformingly mark an unstructured conformal mesh with unflagged planar tetrahedra. To this end, we first introduce the concept of consistent bisection edge. This concept ensures that we can always select the same bisection edge for a given simplex, independently of its dimension. Based on this selection, we propose an element-based marking process that generates unflagged planar tetrahedra. We also check that our marking process is equivalent to the standard face-based marking process proposed by Arnold et al. (2000). Finally, we guarantee that our marking process leads to a conformingly-marked mesh. Accordingly, if we use a restricted version of standard marked bisection to refine the resulting marked mesh, we obtain the optimal number of similarity classes.

4.2.1 Marking edges: strict total order

To mark the mesh edges, we propose a strict total order of the mesh edges. To this end, we use a lexicographic order for the mesh edges that is inherited from the order of the vertices. Specifically, we say that the mesh edge $e_i = (v_{i_1}, v_{i_2})$ has lower global index than the mesh edge $e_j = (v_{j_1}, v_{j_2})$ if $v_{i_1} < v_{j_1}$, or $v_{i_1} = v_{j_1}$ and $v_{i_2} < v_{j_2}$. Note that the proposed lexicographic order is strict and total since it is straightforward to check that is irreflexive, transitive, asymmetric, and connected. Using this lexicographic order, we identify each mesh edge with a unique integer by sorting all the existing edges of the mesh according to the global index criterion.

Algorithm 4.1 Marking as unflagged planar.

input: `Tetrahedron` τ
output: `MarkedTetrahedron` ρ
1: **function** `markTetrahedron`(τ)
2: e_τ = `consistentBisectionEdge`(τ)
3: $(v_0, v_1) = e_\tau$
4: κ_1 = `oppositeFace`(τ, v_0)
5: κ_2 = `oppositeFace`(τ, v_1)
6: e_{κ_1} = `consistentBisectionEdge`(κ_1)
7: e_{κ_2} = `consistentBisectionEdge`(κ_2)
8: $t = P_u$ ▷ Initialize type of tetrahedron
9: $\rho = (\tau, e_\tau, e_{\kappa_1}, e_{\kappa_2}, t)$
10: **return** ρ
11: **end function**

The *consistent bisection edge* of a simplex (tetrahedron or triangle) is the edge with the lowest integer assigned in the edge ordering process. Note that the consistent bisection edge of a simplex is unique because we use a strict total order to characterize it.

4.2.2 Marking tetrahedra: unflagged planar

Using the consistent bisection edge, we propose a marking process of a single tetrahedron that leads to a marked tetrahedron of type P_u, see Algorithm 4.1. The input of the function is a tetrahedron $\tau = (v_0, v_1, v_2, v_3)$ and the output is the corresponding marked tetrahedron ρ. First, we obtain the consistent bisection edge, e_τ, of the tetrahedron, see Line 2. Then, we obtain the opposite triangular faces of the vertices of the bisection edge e_τ, see Lines 4–5. After that, we obtain the corresponding consistent bisection edges e_{κ_1} and e_{κ_2} of κ_1 and κ_2, see Lines 6–7. Finally, we initialize the tetrahedron type, Line 8, as $t = P_u$.

The proposed marking process always generates an unflagged planar tetrahedron. To check it, we need to ensure that the consistent bisection edges selected in Algorithm 4.1 define a triangle of the tetrahedron. Let $\tau = (v_0, v_1, v_2, v_3)$ be a tetrahedron and let us reorder the vertices to have $v_{i_0} < v_{i_1} < v_{i_2} < v_{i_3}$. The consistent bisection edge is $e_\tau = (v_{i_0}, v_{i_1})$ since this is the edge with the lowest indices. The opposite faces to e_τ are $\kappa_1 = (v_{i_1}, v_{i_2}, v_{i_3})$ and $\kappa_2 = (v_{i_0}, v_{i_2}, v_{i_3})$, respectively. For those faces, the consistent bisection edges are $e_{\kappa_1} = (v_{i_0}, v_{i_2})$ and $e_{\kappa_2} = (v_{i_1}, v_{i_2})$, respectively. Since

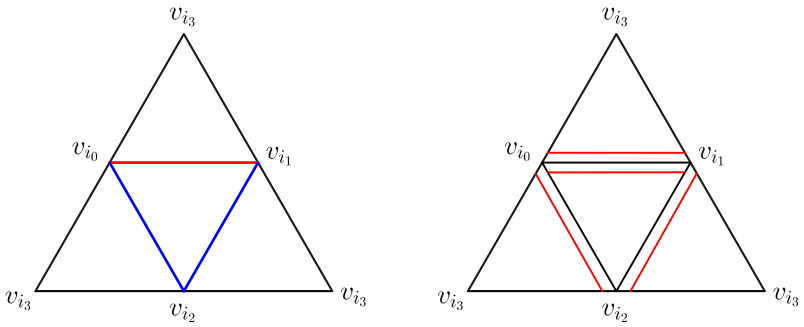


Figure 4.4: Marked tetrahedra with: (a) our element-based marking method; and (b) the standard face-based marking method.

e_τ, e_{κ_1} and e_{κ_2} are connected generating the triangle $(v_{i_0}, v_{i_1}, v_{i_2})$, they define a planar configuration. Figure 4.4(a) shows the obtained marked tetrahedron, where the red edge is the refinement edge and the blue edges are the marked edges corresponding to the non-refinement faces.

We can see that our element-based marking method and the standard face-based one are equivalent, see Figure 4.4. Note that they might not be equivalent since our marking procedure does not exactly proceed as the standard procedure. Specifically, we do not explicitly mark all the triangular faces of a tetrahedron, see Figure 4.4(a). In the standard approach, each face of a tetrahedron has a marked edge that indicates which edge has to be bisected, see red edges in Figure 4.4(b). The refinement edge of the tetrahedron is the only edge that has been marked on both adjacent faces. Thus, after marking all the faces, we obtain that the marked edges of the faces

$$\kappa_1 = (v_{i_0}, v_{i_1}, v_{i_2}), \quad \kappa_2 = (v_{i_0}, v_{i_1}, v_{i_3}),$$
$$\kappa_3 = (v_{i_0}, v_{i_2}, v_{i_3}), \quad \kappa_4 = (v_{i_1}, v_{i_2}, v_{i_3}),$$

are $e_{\kappa_1} = (v_{i_0}, v_{i_1})$, $e_{\kappa_2} = (v_{i_0}, v_{i_1})$, $e_{\kappa_3} = (v_{i_0}, v_{i_2})$ and $e_{\kappa_4} = (v_{i_1}, v_{i_2})$, respectively. Therefore, the refinement edge of τ is $e_\tau = e_{\kappa_1} = e_{\kappa_2}$ and the refinement faces are κ_1 and κ_2. The faces κ_3 and κ_4 are the non-refinement faces and their marked edges are e_{κ_3} and e_{κ_4}, respectively. Thus, all the triangular faces are also marked as an unflagged planar tetrahedron. The edge that is marked from two triangular faces corresponds to the refinement edge of the tetrahedron, see Figure 4.4(b). Thus, the refinement edge and the marked edges obtained with our marking process are equivalent to those obtained with the standard marking process but equipped with our edge ordering. That is, both marking methods generate an equivalent unflagged planar tetrahedron.

Algorithm 4.2 Conformingly marking a tetrahedral mesh.

input: `ConformalMesh` $\mathcal{T}$
output: `ConformalMarkedMesh` $\mathcal{T}'$
1: **function** `markMesh`($\mathcal{T}$)
2: $\mathcal{T}' = \emptyset$
3: **for** $\tau \in \mathcal{T}$ **do**
4: $\rho =$ `markTetrahedron`(τ)
5: $\mathcal{T}' = \mathcal{T}' \cup \rho$
6: **end for**
7: **return** $\mathcal{T}'$
8: **end function**

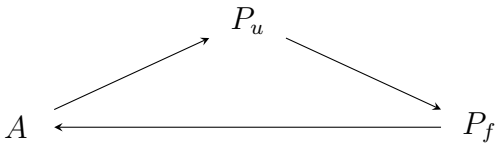


Figure 4.5: Restricted bisection cycle starting on unflagged planar type.

4.2.3 Conformingly marking a mesh

To ensure that we obtain a conformingly-marked mesh, we need that our marking procedure fulfills the sufficient conditions required in Remark 4.1. The first condition is fulfilled since our ordering for mesh edges is strict and total. Furthermore, we know that our element-based marking process is equivalent to the standard face-based marking process. Since both sufficient conditions are fulfilled, we can guarantee that the marking process in Algorithm 4.1 leads to conformingly-marked meshes.

Now, we can detail the method to conformingly mark an unstructured conformal tetrahedral mesh, see Algorithm 4.2. The input is a conformal tetrahedral mesh, $\mathcal{T}$, and the output is a conformingly-marked tetrahedral mesh, $\mathcal{T}'$. We initialize an empty marked mesh and generate a marked tetrahedron ρ for each tetrahedra τ of the mesh $\mathcal{T}$. Then, we insert the marked tetrahedra into the marked mesh $\mathcal{T}'$. Finally, we return the conformingly-marked mesh $\mathcal{T}'$ after marking all the tetrahedra.

Algorithm 4.3 Restriced marked bisection.

input: `MarkedTetrahedron` ρ
output: `MarkedTetrahedron` ρ_1, `MarkedTetrahedron` ρ_2

```
1: function bisectTet(ρ)
2:     t = type(ρ)
3:     if t is P_u then
4:         ρ1, ρ2 = bisectUnflaggedPlanar(ρ)
5:     else if t is P_f then
6:         ρ1, ρ2 = bisectFlaggedPlanar(ρ)
7:     else if t is A then
8:         ρ1, ρ2 = bisectAdjacent(ρ)
9:     end if
10:    return ρ1, ρ2
11: end function
```

4.3 Restricted marked bisection

To bisect our unflagged planar meshes, we consider a restricted version of the standard marked bisection, see Algorithm 4.3. The restricted method bisects a tetrahedron according to its type. Moreover, it only needs to consider the bisection cycle of length three for the tetrahedron types P_u, P_f, and A, see Figure 4.5. In the first case, Line 4, we bisect an unflagged planar tetrahedron. In the second case, Line 6, we bisect a flagged planar tetrahedron. Finally, in the third case, Line 8, we bisect an adjacent tetrahedron.

Figure 4.6 shows how to assign the refinement edge and the marked edges of the children after bisecting a marked tetrahedron of the proposed refinement cycle, according to standard marked bisection. Without loss of generality, we suppose that in all the cases the refinement edge is $e_\tau = (v_0, v_1)$. The vertex ν is the new vertex after the bisection of the edge e_τ. We colored the refinement edge and the marked edges with red and blue, respectively. The first column corresponds to a marked tetrahedron, and the second and third columns correspond to the left and right children, respectively. In rows, we have three different cases. The first row corresponds to the bisection of an unflagged planar tetrahedron to two flagged planar tetrahedra. The second row corresponds to the bisection of a flagged planar tetrahedron to two adjacent tetrahedra. Finally, the third row corresponds to the bisection of an adjacent tetrahedron to two unflagged planar tetrahedra.

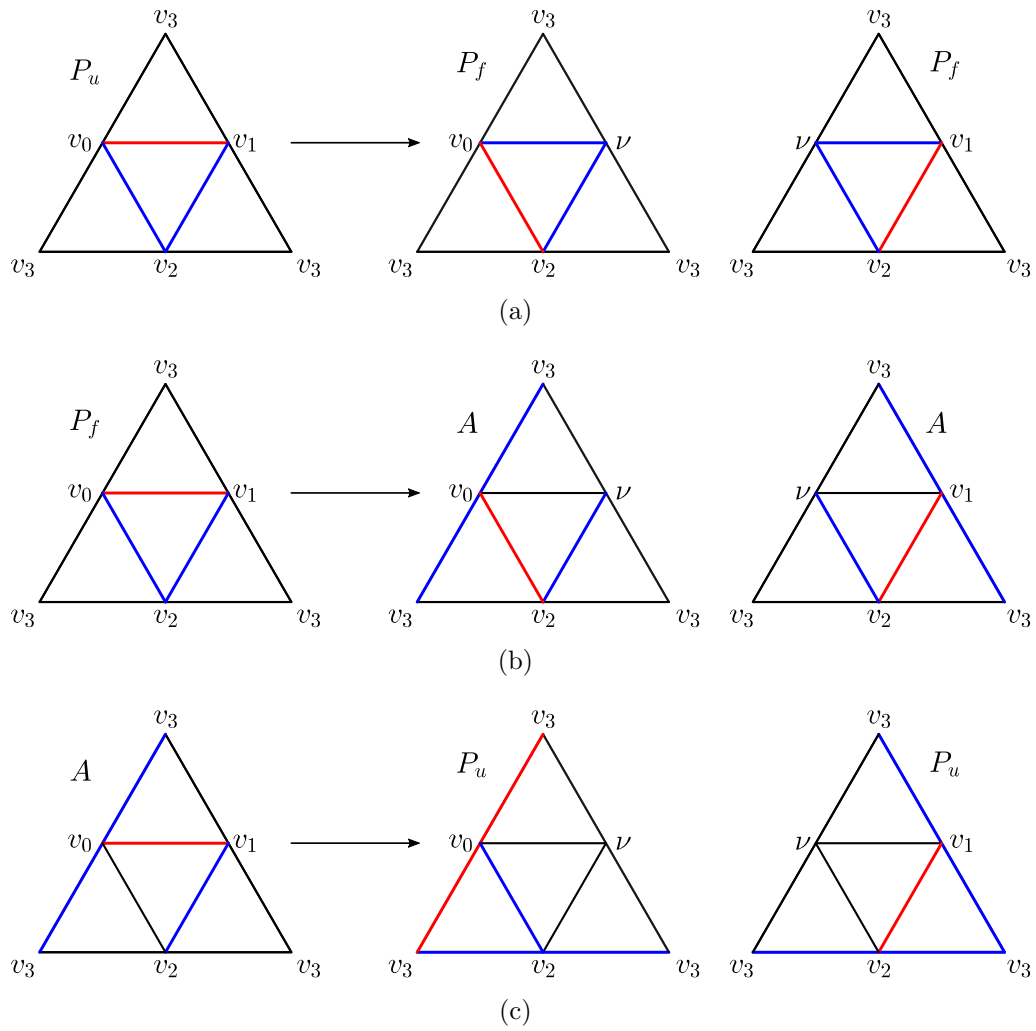


Figure 4.6: Cases for restricted marked bisection: (a) from a P_u to two P_f; (b) from a P_f to two A; and (c) from a A to two P_u.

4.4 Examples

We present several examples to illustrate that our proposed algorithm refines unstructured tetrahedral meshes, generates locally adapted conformal meshes, a finite number of similarity classes, and has a lower-bounded quality. For all the examples, we have computed the shape quality of the mesh elements, see reference Knupp (2001). Then, we plot the minimum and maximum shape quality of the mesh in each refinement step to check that the minimum quality is lower bounded and cycles. Moreover, in the examples where we locally refine the mesh, our code asserts that the mesh is conformal by faces and that Euler's characteristic of the mesh remains

constant.

The results have been obtained on a MacBook Pro with one dual-core Intel Core i5 CPU, at a clock frequency of 2.7GHz, and with a total memory of 16GBytes. As a proof of concept, a mesh refiner has been fully developed in Julia 1.4. The Julia prototype code is sequential (one execution thread), corresponding to the implementation of the method presented in this chapter.

4.4.1 Minimum quality is lower bounded and cycles with uniform refinement

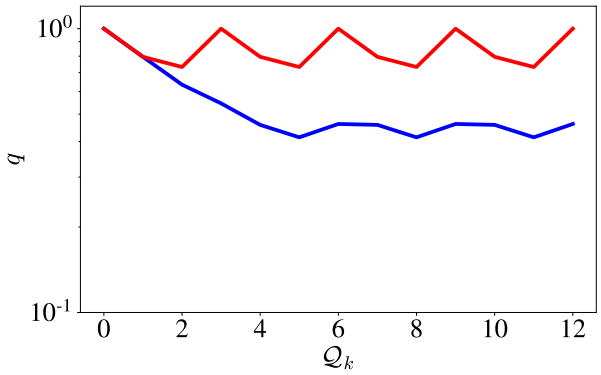
In this example, we show that the minimum quality is lower bounded and cycles. To this end, we uniformly refine a single tetrahedron several times. We denote as $\mathcal{Q}_k$ the obtained mesh after k uniform refinements,

$$\mathcal{Q}_k = \texttt{bisectTetrahedra}(\mathcal{Q}_{k-1}, \mathcal{Q}_{k-1}),$$

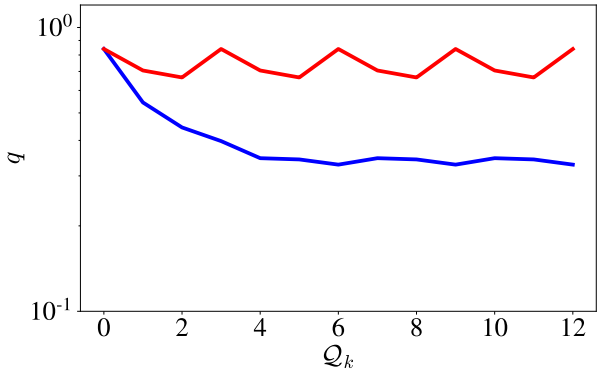
where $\mathcal{Q}_0 = \tau$. Thus, the mesh $\mathcal{Q}_k$ is composed of 2^k tetrahedra and the accumulated number of generated tetrahedra is $2^{k+1} - 1$.

The method needs at least five iterations of uniform refinement to generate 36, different similarity classes. This number of iterations is so since the process only accumulates 31 generated tetrahedra after four successive uniform refinements. Assuming all of these 31 tetrahedra are of a different similarity class, the pigeonhole principle ensures that they cannot correspond to 36 different similarity classes. On the contrary, at the end of iteration five, the accumulated number of generated tetrahedra is 63, greater than 36, and thus, all the similarity classes might be generated. After that iteration, further refinements do not generate new similarity classes, and therefore, the minimum quality remains lower bounded. Moreover, if we perform three additional uniform refinements, we obtain an entire cycle of the quality of length three. To illustrate those quality cycles, we perform a series of additional refinements.

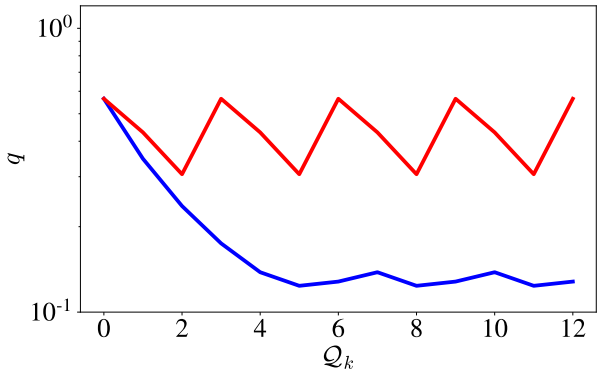
Figure 4.7 plots the evolution of the minimum (blue line) and maximum (red line) qualities during the uniform refinement process. Figures 4.7(a), 4.7(b), and 4.7(c) illustrate the quality of an equilateral, cartesian and a perturbed tetrahedra, respectively. At most, we have to perform five uniform refinements to generate all the similarity classes. Thus, we perform 12 uniform refinements to see how the quality cycles. We can see in Figure 4.7(a), for the most symmetric tetrahedron, how the minimum quality achieves its minimum at iteration five, and then it remains cycling.



(a)

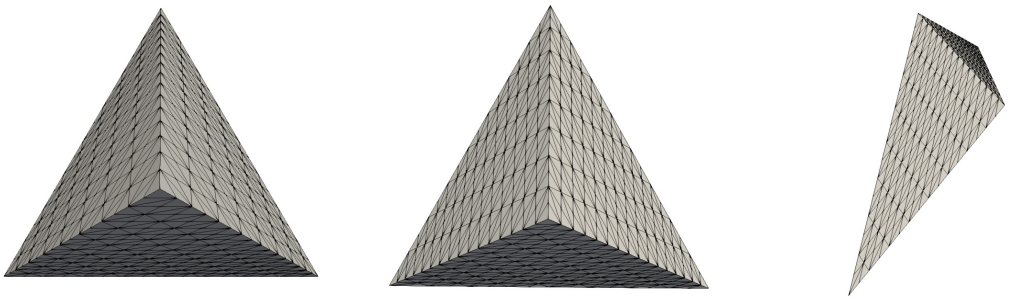


(b)



(c)

Figure 4.7: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iteration: (a) equilateral tetrahedron; (b) cartesian tetrahedron; and (c) random tetrahedron.



(a) (b) (c)

Figure 4.8: Final mesh after 12 iterations of uniform refinement for: (a) equilateral tetrahedron; (b) cartesian tetrahedron; and (c) irregular tetrahedron.

For the cartesian and perturbed tetrahedra, we can see in Figures 4.7(b) and 4.7(c) that we also have to perform five uniform refinements to generate all the similarity classes, achieving the minimum quality of the mesh and start to cycle. In Figures 4.8(a), 4.8(b), and 4.8(c) correspond to the meshes $\mathcal{Q}_{12}$ of the equilateral, cartesian and irregular tetrahedra after 12 uniform refinements.

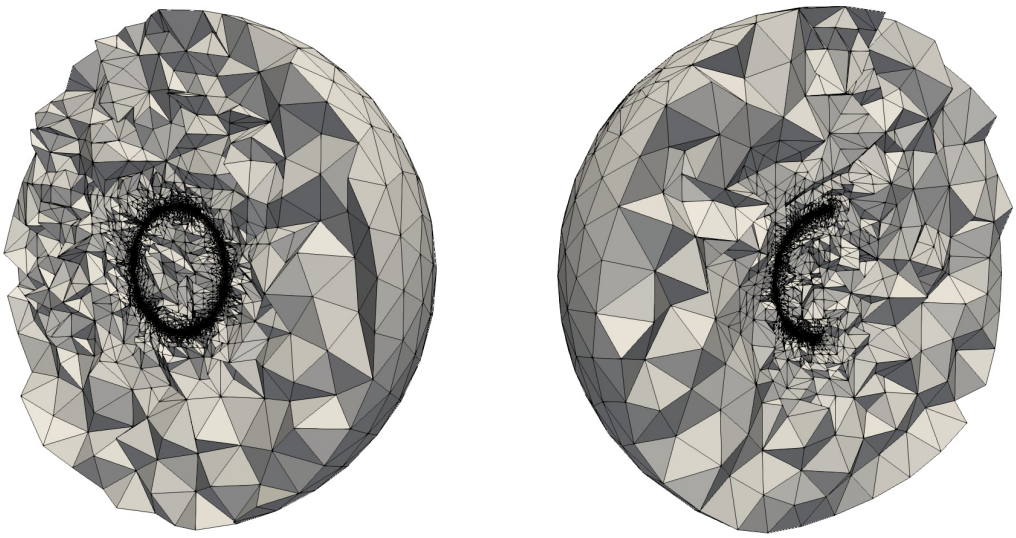
We have generated all the similarity classes for the three tetrahedra, and thus, the minimum mesh quality is achieved. Thus, this example illustrates that the method is stable and the mesh quality does not degenerate during successive refinement.

4.4.2 3D unstructured mesh: locally refining a sphere

This example shows that the proposed refinement scheme can be applied to locally refine unstructured tetrahedral meshes. To this end, we perform the refinement process of Example 3.7.4 with the proposed marked bisection method.

We want to adapt the tetrahedral mesh $\mathcal{T}_0$ to the hemisphere H. At each local refine iteration, we choose the tetrahedra that intersect the hemisphere H as the refinement set. After 40 iterations the mesh $\mathcal{T}_{40}$ is composed by 5806615 tetrahedra and 1045175 vertices. Figures 4.9(a) and 4.9(b) show the $\mathcal{T}_{40}$ sliced with the planes $x = 1/2$ and $y = 1/2$. Figure 4.10 shows how the maximum quality remains constant because it is achieved in each iteration of the local refinement. The minimum quality decreases until its minimum is achieved and then remains constant.

The final mesh is conformal and captures the chosen hemisphere with smaller elements, while it contains larger elements at the exterior boundary.



(a) (b)

Figure 4.9: Slice of the mesh $\mathcal{T}_{40}$ with the plane: (a) $x = 1/2$; and (b) $y = 1/2$.

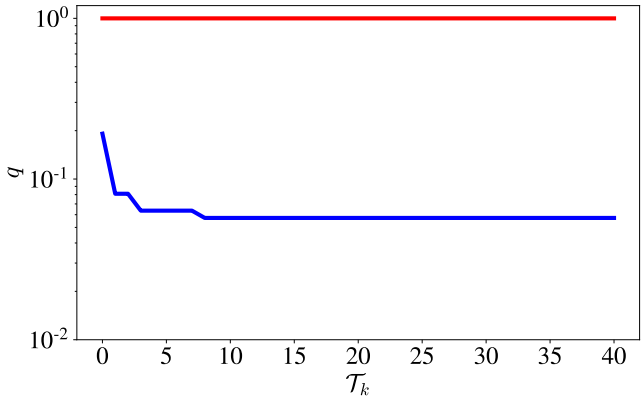
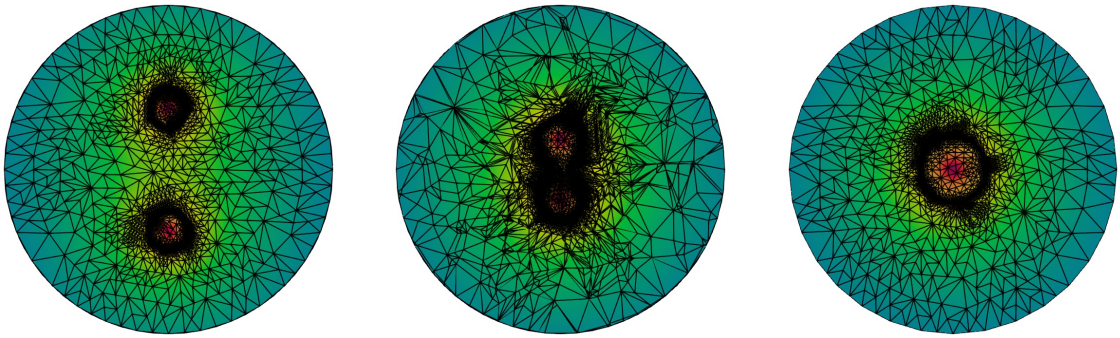


Figure 4.10: Quality of Example 4.4.2: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iterations.

4.4.3 3D space-time mesh: locally refining a iso-potential surface

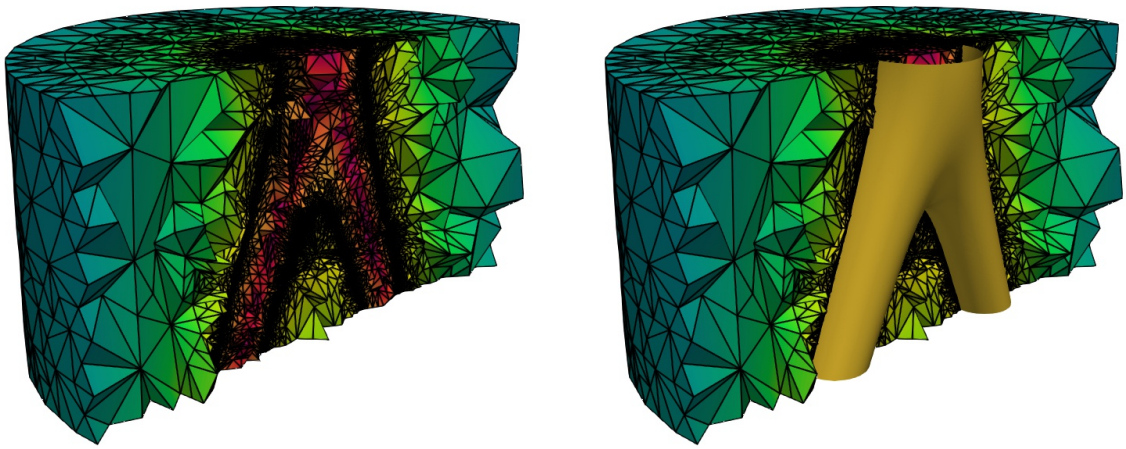
The main goal of this example is to capture a two-dimensional manifold defined by the movement of a one-dimensional manifold. To this end, we perform the refinement process of Example 3.7.5 using the proposed bisection method.

We generate an adapted tetrahedral mesh by locally refining an initial mesh around the manifold. The initial mesh, $\mathcal{T}_0$, is composed of 3781 tetrahedra and 712



(a) (b) (c)

Figure 4.11: Slices of $\mathcal{T}_{50}$ with the plane: (a) $t = 0.0$; (b) $t = 0.5$; and (c) $t = 1.0$.



(a) (b)

Figure 4.12: Slice of the $\mathcal{T}_{50}$ with the plane $x = 1/2$, (a) with, and (b) without the iso-surface.

vertices. After 50 iterations of the local refinement process, the generated mesh $\mathcal{T}_{50}$ has 8356894 tetrahedra and 1504344 vertices. Figures 4.11(a), 4.11(b), and 4.11(c) show a slice of the tetrahedral mesh with the planes $t = 0$, $t = 0.5$, and $t = 1$, respectively. The mesh has been locally refined around the iso-surface and therefore, we have smaller elements near the iso-surface and large elements far from the iso-surface. Figure 4.12 shows a slice of the tetrahedral mesh with the plane $x = 0.5$. We can see how the mesh captures the time evolution of the iso-surface defined by $V(\mathbf{x}, t)$. Figure 4.12(b) shows the iso-surface that is extracted from the space-time

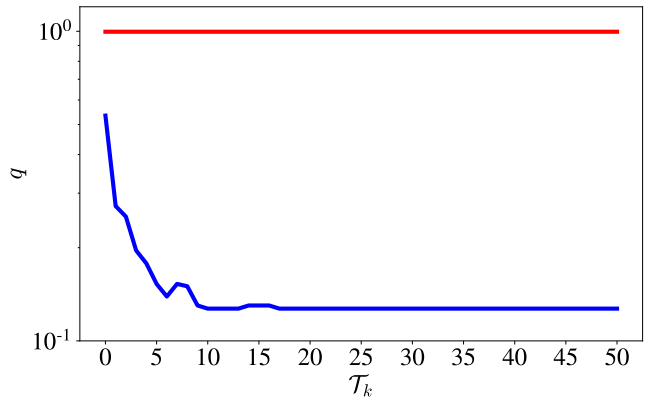


Figure 4.13: Quality of Example 4.4.3: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iterations.

mesh. Figure 4.13 shows how the maximum quality remains constant because it is achieved in each iteration of the local refinement. The minimum quality decreases until its minimum is achieved and then remains constant.

4.5 Concluding remarks

In conclusion, we have shown the first bisection method meeting the bound of 36 similarity classes on three-dimensional unstructured conformal meshes. For these meshes, we have guaranteed that our approach conformingly marks all the tetrahedra as unflagged planar. In this case, marked bisection behaves as the newest vertex bisection, and thus, it features the optimal bound. We have also checked, with our implementation, that the minimum quality cycles for three-dimensional unstructured conformal meshes.

We have answered an open question. Specifically, we have proved that it is possible to mark as unflagged planar all the tetrahedra of an arbitrary three-dimensional unstructured conformal mesh. To explore alternative answers, we will study whether it is possible to mark all the tetrahedra as adjacent or as a mixture of unflagged planar and adjacent elements.

In perspective, our marked bisection allows refining with optimal similarity bound in adaptive applications on three-dimensional complex geometry. The complexity can be handled by the geometrical flexibility of unstructured conformal meshes. On these meshes, our marked bisection meets all the advantages of the newest vertex bisection.

Chapter 5

Newest vertex bisection in n dimensions: reflectivity

The question of whether there is a method to extract a reflection structure on an arbitrary n-dimensional unstructured conformal mesh is still open (Mitchell, 1991, 2017). Answering this question with an explicit algorithm is of major practical interest. Specifically, it would enable applying newest vertex bisection on unstructured meshes and thus, to exploit for the first time the conformity, finiteness, stability, and locality properties on complex n-dimensional geometry. In this chapter, we aim to answer this question and implement the devised method.

To meet this goal, the main contribution is to propose a new marking procedure for n-dimensional unstructured conformal meshes. For these meshes, we guarantee that all the simplices are conformingly-marked as Maubach simplices with a tag equal to n. To this end, we consider three key ingredients. First, we propose to sort the vertices of the simplices with a strict and total order, and we mark all the simplices with Maubach tag equal to n. Second, we prove that this marking procedure extracts a reflected mesh from any n-dimensional unstructured conformal mesh. Third, we prove that this approach ensures strong compatible meshes and thus, enables posterior local refinement with newest vertex bisection.

To illustrate the local refinement application, we implement in n-dimensions the refine to conformity marked bisection in Arnold et al. (2000), but equipped with our marking method. We use the implementation to locally refine unstructured conformal

meshes of different dimensions. Although the conformity, finiteness, stability, and locality are guaranteed, we also illustrate the stability and locality of the resulting meshes.

5.1 Problem and outline of our solution

Given an unstructured simplicial mesh $\mathcal{T}_0$, our problem is to obtain a reflected mesh $\mathcal{T}_1$. Thus, when applying successive iterations of newest vertex bisection, starting on the resulting reflected mesh $\mathcal{T}_1$, we can guarantee an optimal number of similarity classes. Specifically, starting on the reflected mesh $\mathcal{T}_1$, if we locally refine a set of elements $\mathcal{M}$, we obtain a new conformal strongly compatible mesh $\mathcal{T}_2$ with the corresponding elements bisected. The strongly compatible $\mathcal{T}_2$ is suitable for a posterior local refinement. Furthermore, any successive local refinement process has the minimum element quality bounded.

Our solution is to sort the vertices of each simplex and mark the simplex with the tag $d = n$. In this manner, we obtain a reflected mesh in which we can apply Maubach's bisection method.

5.2 Solution: strict total order of vertices leads to reflected meshes

In this section, we first present an algorithm that, for a given conformal unstructured simplicial mesh, reorders the vertices of its elements. Then, we prove that the generated mesh is reflected, and therefore, we can use newest vertex bisection.

5.2.1 Extracting a reflection structure

Given an unstructured simplicial mesh, we detail a method to obtain a reflected mesh, see Algorithm 5.1. The idea of the algorithm is to sort the vertices of each simplex to enforce reflected neighbors.

The input of Algorithm 5.1 is an unstructured conformal simplicial mesh $\mathcal{T}_0$ and the output is a reflected mesh $\mathcal{T}_1$. First, we create an empty mesh $\mathcal{T}_1$ in Line 2. Following, we loop over the simplices of $\mathcal{T}_0$, Line 3, and extract the simplex vertices $(v_0, \ldots, v_n)$, Line 4. Then, we sort the vertices using the order of natural numbers,

Algorithm 5.1 Make a reflected mesh.

input: `Mesh` $\mathcal{T}_0$
output: `ReflectedMesh` $\mathcal{T}_1$

```
1:  function MarkAsReflectedMesh(T_0)
2:      T_1 = ∅
3:      for σ ∈ T_0 do
4:          (v_0, ..., v_n) = σ
5:          (v_{i_0}, ..., v_{i_n}) = sort(v_0, ..., v_n)
6:          σ̄ = ([v_{i_0}], ..., [v_{i_n}])
7:          d = n
8:          l = 0
9:          μ = (σ̄, d, l)
10:         T_1 = T_1 ∪ μ
11:     end for
12:     return T_1
13: end function
```

which is a strict total order, see Line 5. After that, we perform a mapping between the sorted list of vertices and a sorted list of multi-ids of length one, and create a new simplex, $\bar{\sigma}$, see Line 6. Since we have not bisected any edge yet, all the multi-ids are of length one. In Lines 7 and 8, we set the tag $d = n$ and descendant level $l = 0$, respectively. According to Remark 2.6, we can also use the tag $d = n - 1$. Then, we set the Maubach simplex $\mu = (\bar{\sigma}, d, l)$, see Line 9, and add it to $\mathcal{T}_1$, see Line 10. A Maubach simplex is a 3-tuple $\mu = (\bar{\sigma}, d, l)$, where $\bar{\sigma}$ is an equivalent simplex, but it is reordered to properly contribute to a reflected mesh, d is a Maubach integer tag, and l is the descendant level. Finally, we return the reflected mesh $\mathcal{T}_1$ in Line 12.

5.2.2 Sufficient condition for reflectivity

In this subsection, we prove that if we have a strict total order of mesh vertices and that all the simplices have their vertices ordered accordingly to it, then the mesh is reflected. Then, we use this result to prove our Algorithm 5.1 generates reflected meshes. Before that, we present some definitions and functions that we will use in the proof. Note that there are functions with the same name but with different input and output. In this section, we consider a given strict total order, $\ll$, of mesh vertices.

- For a given simplex $(s_0, \ldots, s_n)$, the function `set` returns the set of vertices

$\{s_0, \ldots, s_n\}$:

$$\begin{aligned} \texttt{set} &: \texttt{Simplex} \longrightarrow \texttt{Set} \\ \texttt{set}\,(s_0, \ldots, s_n) &= \{s_0, \ldots, s_n\}. \end{aligned} \tag{5.1}$$

- For a given tuple $(s_0, \ldots, s_n)$, the constructor function `Simplex` returns the simplex $(s_0, \ldots, s_n)$:

$$\begin{aligned} \texttt{Simplex} &: \texttt{Tuple} \longrightarrow \texttt{Simplex} \\ \texttt{Simplex}\,(s_0, \ldots, s_n) &= \texttt{Simplex}(s_0, \ldots, s_n). \end{aligned} \tag{5.2}$$

- For a given set of vertices $\{s_0, \ldots, s_n\}$, the function `sort` returns the sorted tuple $(s_{i_0}, \ldots, s_{i_n})$, where $i_0, i_1, \ldots, i_n$ is a permutation such that $s_{i_0} \ll \ldots \ll s_{i_n}$:

$$\begin{aligned} \texttt{sort} &: \texttt{Set} \longrightarrow \texttt{Tuple} \\ \texttt{sort}\,\{s_0, \ldots, s_n\} &= (s_{i_0}, \ldots, s_{i_n}). \end{aligned} \tag{5.3}$$

- For a given set of vertices $\{s_0, \ldots, s_n\}$, the function `simplex` returns a sorted simplex $(s_{i_0}, \ldots, s_{i_n})$, where $i_0, i_1, \ldots, i_n$ is a permutation such that $s_{i_0} \ll \ldots \ll s_{i_n}$:

$$\begin{aligned} \texttt{simplex} &: \texttt{Set} \longrightarrow \texttt{Simplex} \\ \texttt{simplex}\,\{s_0, \ldots, s_n\} &= \texttt{Simplex} \circ \texttt{sort}\,\{s_0, \ldots, s_n\}. \end{aligned} \tag{5.4}$$

- For a given simplex $(s_0, \ldots, s_n)$, the function `sort` returns a sorted simplex $(s_{i_0}, \ldots, s_{i_n})$, where $i_0, i_1, \ldots, i_n$ is a permutation such that $s_{i_0} \ll \ldots \ll s_{i_n}$:

$$\begin{aligned} \texttt{sort} &: \texttt{Simplex} \longrightarrow \texttt{Simplex} \\ \texttt{sort}\,(s_0, \ldots, s_n) &= \texttt{simplex} \circ \texttt{set}\,(s_0, \ldots, s_n). \end{aligned} \tag{5.5}$$

- For two given simplices $(s_0, \ldots, s_n)$ and $(t_0, \ldots, t_m)$, the function `+` returns the concatenation simplex $(s_0, \ldots, s_n, t_0, \ldots, t_m)$:

$$\begin{aligned} \texttt{+} &: \texttt{Simplex} \times \texttt{Simplex} \longrightarrow \texttt{Simplex} \\ \texttt{+}\,(s_0, \ldots, s_n)\,(t_0, \ldots, t_m) &= (s_0, \ldots, s_n, t_0, \ldots, t_m). \end{aligned} \tag{5.6}$$

Note that if σ is already sorted the function `sort` of Equation (5.5) holds that

$$\texttt{sort}(\sigma) = \sigma.$$

Let $\sigma = (v_0, \ldots, v_n)$ be a sorted simplex and let v be a vertex of σ. Consider the decomposition of σ as follows

$$\sigma = (v_0, \ldots, v_n) = (v_0, \ldots, v_{k-1}, v, v_{k+1}, \ldots, v_n), \tag{5.7}$$

where k is such that $v_k = v$. That is, for a given simplex σ and a vertex v of σ, we have that $\sigma = l \texttt{ + } (v) \texttt{ + } r$, where $l = (v_0, \ldots, v_{k-1})$ and $r = (v_{k+1}, \ldots, v_n)$ hold that $l_0 \ll \ldots \ll l_{k-1} \ll v \ll r_{k+1} \ll \ldots \ll r_n$. The interpretation of that decomposition is that for a given sorted simplex σ and a vertex v, we can split the simplex into two sub-simplices composed of the vertices smaller than v, and the vertices greater than v.

For the strict total order $\ll$ of mesh vertices, we consider the two following definitions:

Definition 5.1 (Sorted simplex)**.** A simplex $(v_0, \ldots, v_n)$ is sorted if

$$v_0 \ll \ldots \ll v_n.$$

Definition 5.2 (Sorted simplicial mesh)**.** A simplicial mesh is sorted if all the mesh simplices are sorted.

Theorem 5.1 (Reflectivity sufficient condition)**.** *Let $\mathcal{T}$ be a conformal n-dimensional simplicial mesh. If the mesh is sorted and all the Maubach simplices have tag equal to n, then the mesh $\mathcal{T}$ is reflected.*

Proof. We have to check that all pairs of adjacent neighboring simplices $\bar{\sigma}_1$ and $\bar{\sigma}_2$ of $\mathcal{T}$ are reflected neighbors. That is, both simplices have the same tag, and the shared face $\bar{\kappa} = \texttt{simplex}(\texttt{set}(\bar{\sigma}_1) \cap \texttt{set}(\bar{\sigma}_2))$ is ordered in the same manner from $\bar{\sigma}_1$ and $\bar{\sigma}_2$.

By hypothesis all the Maubach simplices have the same tag $d = n$. Thus, we only need to proof that the shared face has the same order of vertices inside $\bar{\sigma}_1$ and $\bar{\sigma}_2$.

Since $\mathcal{T}$ is a sorted simplicial mesh, we have that $\bar{\sigma}_1$ and $\bar{\sigma}_2$ are sorted simplices by Definition 5.2. Therefore, we have that

$$\begin{aligned} \bar{\sigma}_1 &= (v_0, \ldots, v_n) \text{ with } v_0 \ll \ldots \ll v_n, \\ \bar{\sigma}_2 &= (w_0, \ldots, w_n) \text{ with } w_0 \ll \ldots \ll w_n, \end{aligned}$$

according to Definition 5.1. Moreover, let

$$\bar{\kappa} = \texttt{simplex}(\texttt{set}(\bar{\sigma}_1) \cap \texttt{set}(\bar{\sigma}_2)) = (u_0, \ldots, u_{n-1})$$

be the shared sorted face between $\bar{\sigma}_1$ and $\bar{\sigma}_2$. Since $\bar{\sigma}_1$ and $\bar{\sigma}_2$ are neighbors, there is only one v in $\texttt{set}(\bar{\sigma}_1)$ that is not in $\texttt{set}(\bar{\sigma}_2)$, and there is only one w in $\texttt{set}(\bar{\sigma}_2)$ that is not $\texttt{set}(\bar{\sigma}_1)$.

Now, we can consider the decomposition of $\bar{\sigma}_1$ and $\bar{\sigma}_2$ as in Equation (5.7)

$$\bar{\sigma}_1 = l_1 \texttt{ + } (v) \texttt{ + } r_1,$$
$$\bar{\sigma}_2 = l_2 \texttt{ + } (w) \texttt{ + } r_2.$$

Next, we prove that $\bar{\kappa}$ is equal to $l_1 \texttt{ + } r_1$. Since the vertices of $\bar{\kappa}$ are sorted, we have that $\bar{\kappa} = \texttt{sort}(\bar{\kappa})$. Applying that `sort` is equal to $\texttt{simplex} \circ \texttt{set}$, Equation (5.4), we have that

$$\texttt{sort}(\bar{\kappa}) = \texttt{simplex}(\texttt{set}(\bar{\kappa})).$$

Since $\texttt{set}(\bar{\kappa}) = \texttt{set}(l_1) \cup \texttt{set}(r_1)$ by Lemma A.2, we obtain that

$$\texttt{simplex}(\texttt{set}(\bar{\kappa})) = \texttt{simplex}(\texttt{set}(l_1) \cup \texttt{set}(r_1)).$$

Finally,

$$\texttt{simplex}(\texttt{set}(l_1) \cup \texttt{set}(r_1)) = l_1 \texttt{ + } r_1$$

by Lemma A.3.

We have obtained that $\bar{\kappa} = l_1 \texttt{ + } r_1$. Similarly, we can repeat the argument with the simplex $\bar{\sigma}_2$ to obtain $\bar{\kappa} = l_2 \texttt{ + } r_2$. Accordingly, we obtain $l_1 \texttt{ + } r_1 = l_2 \texttt{ + } r_2$, and therefore the shared face $\bar{\kappa}$ in $\bar{\sigma}_1$ and $\bar{\sigma}_2$ appears with the same order in both simplices.

Thus, we have proved that any pair of simplices $\bar{\sigma}_1$ and $\bar{\sigma}_2$ of $\mathcal{T}$ are reflected neighbors. Therefore, the sorted simplicial mesh $\mathcal{T}$ is a reflected mesh as we wanted to see. □

5.2.3 Newest vertex bisection on unstructured meshes: extracting reflectivity and strong compatibility

Next, we guarantee that the proposed marking procedure enables local refinement with the newest vertex bisection. To this end, we use the theorem of the reflectivity sufficient condition to ensure that the marking procedure leads to reflected meshes. This reflectivity guarantees that the resulting meshes feature strong compatibility, thus, ensuring the applicability of the newest vertex bisection.

Corollary 5.1 (Marking as reflected)**.** Let $\mathcal{T}_0$ be a conformal simplicial mesh, and $\mathcal{T}_1$ the obtained mesh after the application of Algorithm 5.1 to $\mathcal{T}_0$. Then, $\mathcal{T}_1$ is a reflected mesh.

Proof. We have to check that the mesh $\mathcal{T}_1$ is reflected. To this end, we will check that all the simplices in $\mathcal{T}_1$ are sorted simplices and, therefore, that $\mathcal{T}_1$ is a sorted simplicial mesh. Moreover, we will check that all the Maubach simplices have tag $d = n$. Then, by Theorem 5.1, $\mathcal{T}_1$ is reflected.

By construction, Algorithm 5.1 sorts all the simplices with the order of the natural numbers, which is strict and total. Accordingly, all the simplices in $\mathcal{T}_1$ are sorted simplices according to Definition 5.1. Thus, the mesh $\mathcal{T}_1$ is a sorted simplicial mesh according to Definition 5.2. Since Algorithm 5.1 assigns to all the Maubach simplices the same tag $d = n$, Theorem 5.1 ensures that the mesh $\mathcal{T}_1$ is reflected. □

Corollary 5.2 (Strong compatibility)**.** Let $\mathcal{T}_0$ be a conformal n-simplicial mesh, and let $\mathcal{T}_1$ be the mesh obtained with Algorithm 5.1. Then, $\mathcal{T}_1$ is suitable to use newest vertex bisection for local refinement.

Proof. The mesh $\mathcal{T}_1$ is reflected by Corollary 5.1 and, therefore, by Remark 2.6, is strongly compatible. By Theorem 5.1 of Stevenson (2008), if we locally refine $\mathcal{T}_1$ using newest vertex bisection, the obtained mesh is conformal and strongly compatible. □

5.2.4 Number of iterations to obtain all the similarity classes

We want to calculate the minimum number of uniform refinements required to generate all the similarity classes with newest vertex bisection. To do that, we consider uniform refinements in order to generate the maximum number of simplices per iteration. Thus, let $\mathcal{Q}_0^\sigma = \sigma$ and

$$\mathcal{Q}_k^\sigma = \texttt{bisectSimplices}(\mathcal{Q}_{k-1}^\sigma, \mathcal{Q}_{k-1}^\sigma)$$

the obtained mesh after performing k uniform refinements, a mesh $\mathcal{Q}_i^\sigma$ that is composed of $\#\left(\mathcal{Q}_i^\sigma\right) = 2^i$ simplices. Considering all the meshes $\mathcal{Q}_0^\sigma, \ldots \mathcal{Q}_k^\sigma$, we have at most

$$\sum_{i=0}^{k} \#\left(\mathcal{Q}_i^\sigma\right) = \sum_{i=0}^{k} 2^i = 2^{k+1} - 1 \tag{5.8}$$

different simplices.

Recall that newest vertex bisection generates at most $M_n = nn!2^{n-2}$ similarity classes, see Theorem 4.5 of Arnold et al. (2000). We remark that the number of generated similarity classes is an upper bound. That is, the method can generate

K similarity classes, where $K \leq M_n$. Thus, in the case that the method generates M_n similarity classes, it needs to refine at least k times uniformly to generate M_n similarity classes, where k holds that

$$2^{k+1} - 1 \geq nn!2^{n-2}.$$

Thus, k has to fulfill that

$$2^{k+1} \geq 1 + nn!2^{n-2}.$$

Applying logarithm in both sides, we obtain that

$$k \geq \lceil \log_2(1 + nn!2^{n-2}) \rceil - 1.$$

Assuming all of these $2^{k+1} - 1$ simplices are of a different similarity class, the pigeonhole principle ensures that they cannot correspond to M_n different similarity classes. On the contrary, at the end of iteration k, the accumulated number of generated simplices is $2^{k+1} - 1$, greater than M_n.

Remark 5.1 (Number of bisection iterations)**.** With this reasoning, we show that we need at least $\lceil \log_2(1+nn!2^{n-2}) \rceil - 1$ iterations to get all the similarity classes. We do not ensure that we get all the similarity classes. Nevertheless, it is a lower bound over the necessary uniform refinement to generate M_n similarity classes.

5.3 Examples

We present four examples where we refine different meshes with our bisection method. In the first example, we illustrate the number of similarity classes for different simplices and the number of iterations to obtain them. Next, we locally refine a hypersphere in different dimensions to study the evolution of the constant C controlling the growth of the conformal closure (Binev et al., 2004; Stevenson, 2008). In the third example, we locally refine 4-dimensional unstructured meshes of a hypersphere to locally adapt a smaller hypersphere. Finally, in the fourth example, we locally refine a hypercylinder to capture the time-evolution of a potential.

All the results have been obtained on a computer with an Intel Core i9-9900K, with a clock frequency of 3.6GHz, and total memory of 8TBytes. As a proof of concept, a mesh refiner has been fully developed in Julia 1.4.2. The Julia prototype code is sequential (one execution thread), corresponding to the implementation of the method presented in this chapter. All the unstructured initial meshes are generated with the `distmesh` algorithm (Persson and Strang, 2004).

Table 5.1: Number of generated similarity classes by newest vertex bisection.

Dimension	Equilateral	Cartesian	Kuhn	Irregular	M_n
2	3	4	1	4	4
3	36	27	3	36	36
4	52	58	4	384	384
5	2079	312	5	4800	4800

Table 5.2: Number of uniform refinements to generate all the similarity classes.

Dimension	Equilateral	Cartesian	Kuhn	Irregular	Minimum for irregular
2	2	2	0	2	2
3	7	5	2	7	5
4	10	10	3	10	8
5	15	15	4	17	12

5.3.1 Iterations versus number of similarity classes

We present the number of similarity classes and the minimum number of uniform refinements for different simplices and dimensions. We uniformly refine an equilateral simplex, a Cartesian simplex, a Kuhn simplex, and an irregular simplex for dimensions 2, 3, 4, and 5. The number of similarity classes and the needed uniform refinements are detailed in Table 5.1 and Table 5.2, respectively.

The equilateral simplex has all its edges of the same length, the Cartesian simplex has vertices determined by the origin and the canonical vectors e_i, and the Kuhn simplex, which is one of the simplices obtained after dividing a hypercube with Coxeter-Freudenthal-Kuhn algorithm. Finally, the irregular simplex has all its edges with different lengths. To obtain all the similarity classes, we used the quality of the simplices as a proxy, assigning an obtained shape quality to a similarity class. We refined the initial simplices and their descendants uniformly until the method does not generates more similarity classes.

Table 5.1 shows the number of obtained similarity classes for each case. The equilateral and Cartesian simplex have fewer similarity classes than M_n. That is because they have geometric symmetries, and thus Maubach's bisection generates less similarity classes than M_n. On the other hand, the Kuhn simplex is the one that generates the minimum number of similarity classes. That is because newest

vertex bisection achieves its optimal number of similarity classes with structured meshes, which are fully composed of Khun simplices. Finally, the irregular simplex generates the maximum number of similarity classes due to its lack of symmetry. That is, it generates the predicted maximum number of similarity classes. As dimension increases, the number of similarity classes also increases in all the cases.

Table 5.2 shows the number of uniform refinements performed to generate the similarity classes of Table 5.1, and the a lower bound over the number of uniform refinements to generate M_n similarity classes. We see that the equilateral, the Cartesian, and the irregular simplices exceed the number of minimum uniform refinements to generate the similarity classes of Table 5.1. Moreover, the number of uniform refinements of the equilateral and Cartesian simplices is smaller than the irregular simplex. For the Kuhn simplex, we can see that the number of uniform refinements to achieve the generated number of similarity classes is $n-1$, except in the 2D case. That is because the initial simplex is the unique similarity class. Generally, when the number of similarity classes becomes larger, we need to perform more uniform refinements to generate them.

5.3.2 Conformal closure

In this example, we illustrate that our refinement method fulfills the bounds presented in Theorem 6.1 of Stevenson (2008) under successive local refinements. To do that, we adapt the mesh of an n-dimensional closed ball of radius 1 to capture a hypersphere of radius 10^{-3}. We perform this procedure for dimensions 2, 3, 4, and 5.

Let $\bar{B}^n$ be the n-dimensional closed ball of radius 1 centered at the origin, defined as the points such that $\|\mathbf{x}\| \leq 1$. We approximate the domains $\bar{B}^2$, $\bar{B}^3$, $\bar{B}^4$, and $\bar{B}^5$ with the simplicial meshes $\mathcal{T}_0^2$, $\mathcal{T}_0^3$, $\mathcal{T}_0^4$, and $\mathcal{T}_0^5$, respectively. All the meshes have been generated using `distmesh` (Persson and Strang, 2004). In all the cases, the meshes have the same edge length, $h_e = 0.5$. We perform 10 iteration of local refinement, generating the simplicial meshes $\mathcal{T}_{10}^2$, $\mathcal{T}_{10}^3$, $\mathcal{T}_{10}^4$, and $\mathcal{T}_{10}^5$.

Let

$$\mathcal{T}_{k+1} = \texttt{localRefine}\,(\mathcal{T}_k, \mathcal{M}_k)$$

be the obtained mesh, $\mathcal{M}_k$ be the set of marked simplices to be bisected at iteration k, and $\mathcal{M}_{0,k} = \mathcal{M}_0 \cup \ldots \cup \mathcal{M}_{k-1}$ be the union of the sets marked simplices to be refined until the k-th iteration, where $1 \leq k \leq 10$. We use Theorem 6.1 of Stevenson (2008) to study the evolution of C during the 10 iterations of local refine for each

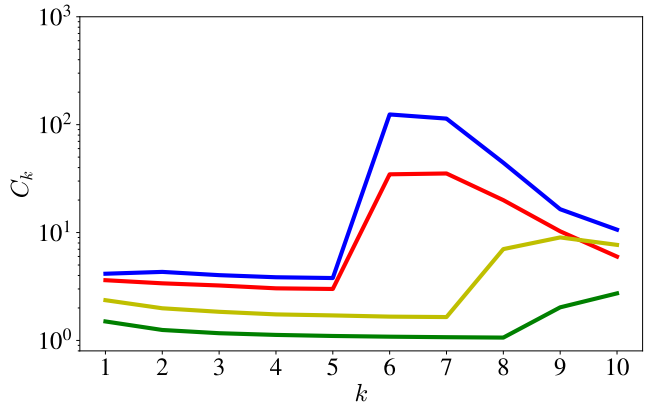


Figure 5.1: We show the evolution of C_k for the meshes in 2D (blue line), 3D (red line), 4D (yellow line), and 5D (green line) during the 10 iterations of local refinement.

mesh. We approximate C in each iteration with C_k, approximating Equation (2.1) as

$$C_k = \frac{\#\left(\mathcal{T}_{k+1}\right) - \#\left(\mathcal{T}_0\right)}{\#\left(\mathcal{M}_{0,k}\right)}.$$

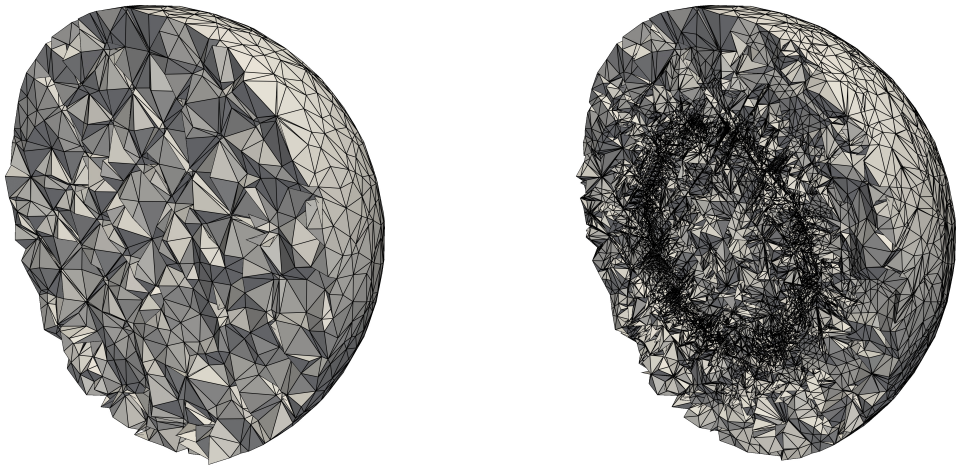
Figure 5.1 shows the evolution of C_k for dimensions 2, 3, 4, and 5. For dimensions 2 and 3, we see that C_k increases at iteration 6 and then, it tends to decrease. For dimension 4, C_k starts to increase at iteration 7 and at iteration 9 it decreases. Finally, for dimension 5 we see that C_k starts to increase at iteration 8.

5.3.3 4D unstructured mesh: locally refining a hypersphere

This example shows that the proposed refinement scheme can be applied to locally refine unstructured simplicial meshes. To this end, we perform the refinement process of Example 3.7.4 with the proposed newest vertex bisection method.

The initial mesh has an edge length of 0.15 and is composed of 218229 pentatopes and 10362 vertices. Figure 5.2(a) shows a slice of $\mathcal{T}_0$ with the hyperplane $t = 0.0$. After 7 iterations of the refinement process, the obtained mesh $\mathcal{T}_7$ is composed of 73121697 pentatopes and 3568201 vertices. We slice $\mathcal{T}_7$ with the hyperplane $t = 0.0$ to obtain the 3D tetrahedral representation depicted in Figure 5.2(b). We can see how the mesh is locally refined capturing the inner hypersphere.

Figure 5.3 shows the evolution of the shape quality during the refinement process. We see that the maximum quality remains constant during the refinement process. The minimum quality of the mesh decreases but does not achieve its minimum since



(a) (b)

Figure 5.2: Slice of the 4-simplicial mesh of a hypersphere with the hyperplane $t = 0$: (a) initial mesh; and (b) locally adapted mesh $\mathcal{T}_7$.

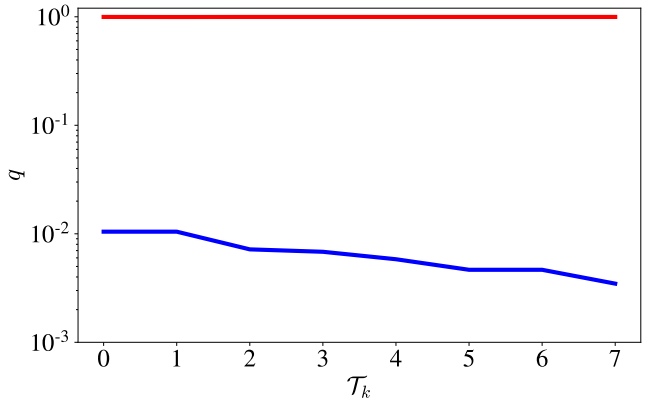


Figure 5.3: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iteration.

we need to perform more local refinements. Figure 5.4 shows the evolution of C during the refinement process and how it decreases after the third local refinement step.

The obtained results illustrate that the proposed bisection algorithm can refine locally unstructured simplicial meshes and generates conformal meshes.

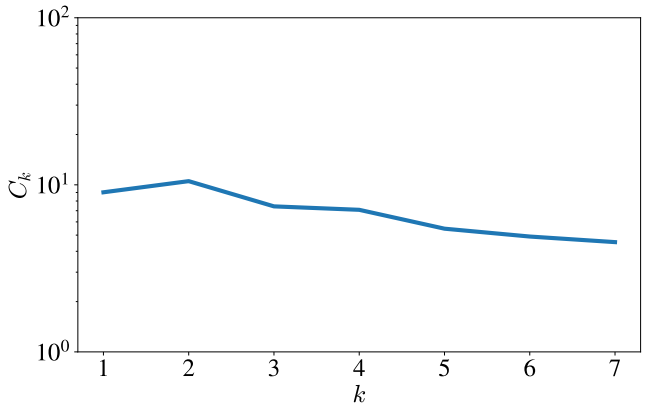


Figure 5.4: Evolution of C_k during the local refinement process.

5.3.4 4D Space-time local refinement: time evolution of a potential

The main goal of this example is to capture a three-dimensional manifold defined by the movement of a two-dimensional manifold. To this end, we perform the refinement process of Example 3.7.5 using the proposed bisection method. curvature.

After 22 iterations of the local refinement process, the generated mesh $\mathcal{T}_{22}$ has 35686663 pentatopes, 1773294 vertices, and 23221152 edges. To visualize the obtained mesh, we sliced it with a hyperplane to obtain a 3D tetrahedral representation. Figures 5.5(a), 5.5(c), and 5.5(e) show a slice of the pentatopic mesh with the hyperplanes $t = 0$, $t = 0.5$, and $t = 1$, respectively. The mesh has been locally refined around the isosurface and therefore, we have smaller elements near the isosurface and large elements far from the isosurface. Figures 5.5(b), 5.5(d), and 5.5(f) show the isosurface that is extracted from the mesh. These slices show that the isosurface starts at two connected components and then join into one connected component. Figure 5.6(a) shows a slice of the pentatopic mesh with the hyperplane $x = 0.5$, generating the space-time mesh (z, y, t). We can see how the mesh captures the time evolution of the isosurface defined by $V(\mathbf{x}, t)$. Figure 5.6(b) shows the isosurface that is extracted from the space-time mesh.

Figure 5.7 shows the evolution of the maximum and minimum quality of the mesh during the local refinement process. We see that the maximum quality remains constant during the refinement process. The minimum quality of the mesh decreases until iteration 16, and then it stabilizes since, in posterior local refinements, the

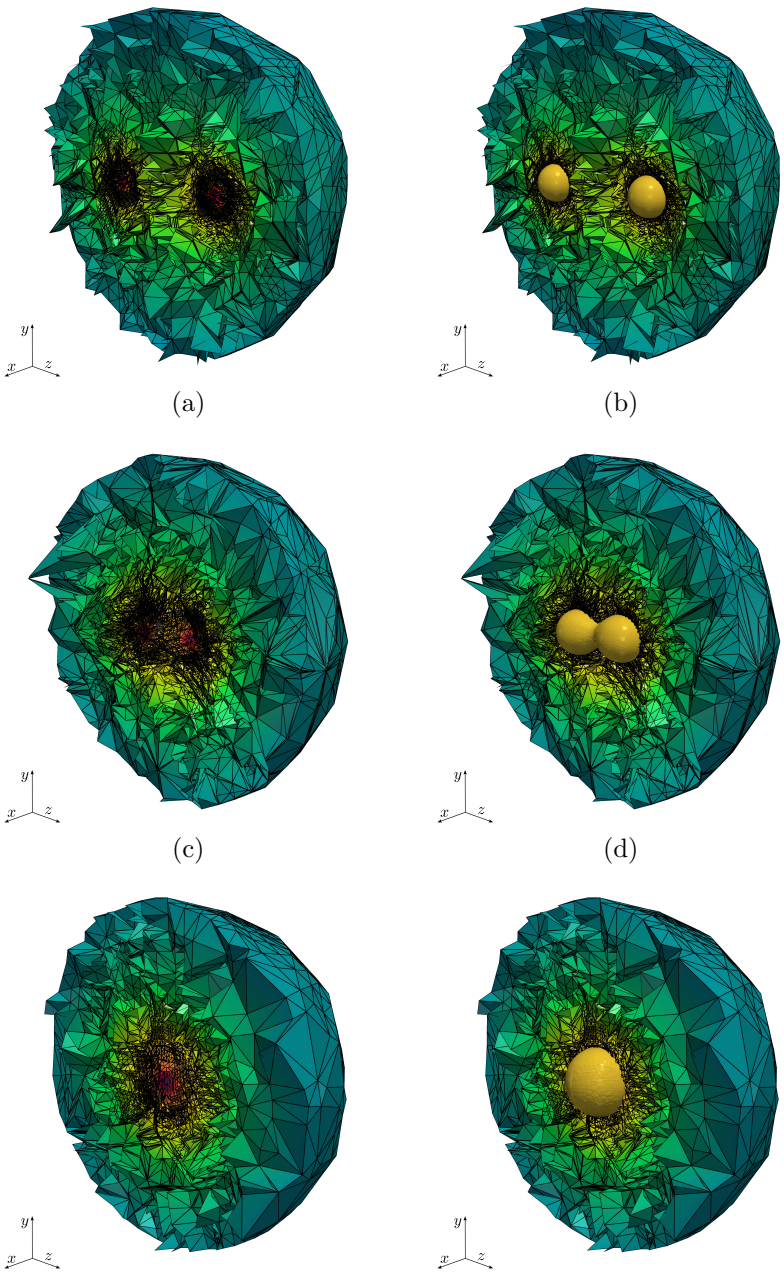
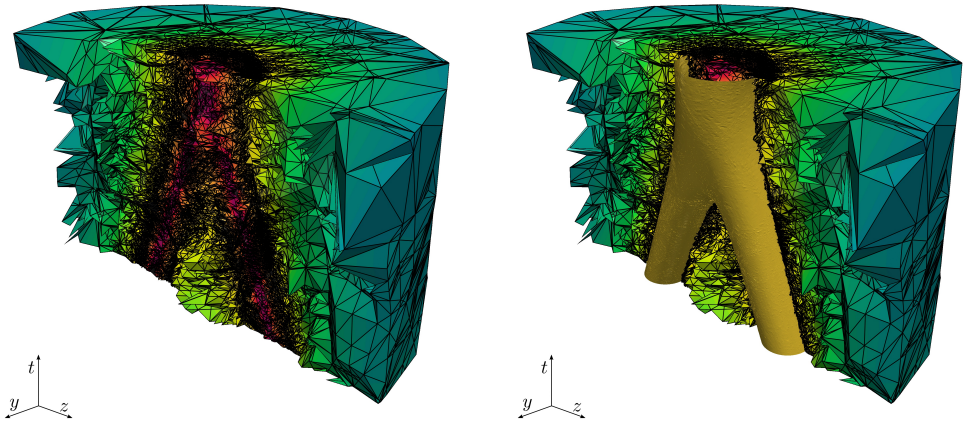


Figure 5.5: Different slices in time of $\mathcal{T}_{22}$ are presented to illustrate how the isosurface has been captured. In rows, slice with $t = 0.0$ (a) and (b), slice with $t = 0.5$ (c) and (d), and slice with $t = 1.0$ (e) and (f). In columns, slice of the mesh (a), (c) and (e), and countour of the isosurface (b), (d) and (f).

minimum mesh quality is achieved. Figure 5.8 shows the evolution of C_k during the local refinement process. We see how C_k decreases after the third local refinement



(a) (b)

Figure 5.6: Slice with the hyperplane $x = 1/2$. In Figures (a) and (b) we obtain the 3D space-time mesh (z, y, t), where we can see the time evolution of the isosurface defined by the gravitational potential. We can see how the mesh is adapted to capture the movement of the two particles.

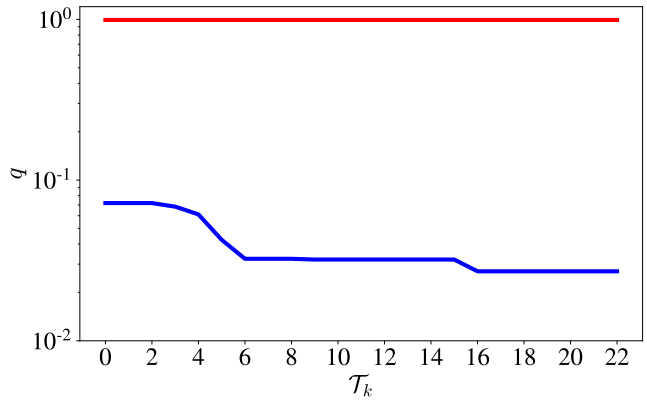


Figure 5.7: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iteration.

step. In the last iterations, C_k is approximately ten, and this means that for each marked simplex, we had to refine ten additional simplices to obtain a conformal mesh.

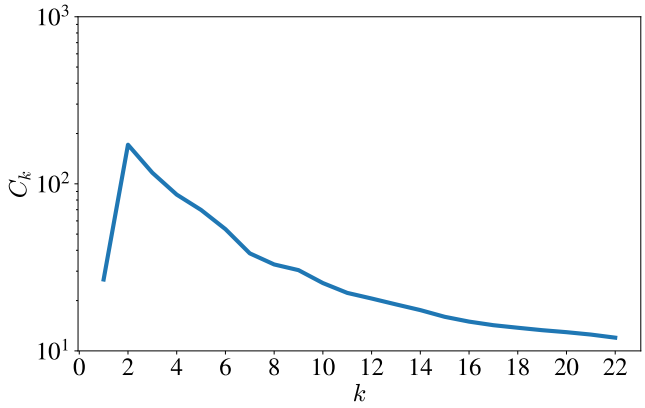


Figure 5.8: Evolution of C_k during the local refinement process.

5.4 Concluding remarks

In conclusion, we have proposed and implemented the first n-dimensional bisection method for n-simplicial unstructured conformal meshes that can be applied locally, generating conformal meshes in a finite number of steps, see references Maubach (1995) and Stevenson (2008), that leads to at most M_n similarity classes, see Arnold et al. (2000), and that its conformal closure is optimal for finite element methods (Binev et al., 2004; Stevenson, 2008). For these meshes, we have guaranteed that our approach generates reflected meshes. In this case, the generated meshes are suitable for using newest vertex bisection with local refinement.

We have answered an open question. Specifically, we have proved that it is possible to reorder an arbitrary n-simplicial unstructured conformal mesh such that the obtained mesh is reflected.

In perspective, our bisection method enables adaptive applications on n-dimensional complex geometry. In this case, the complexity of the geometry would be handled by the flexibility of unstructured conformal meshes. Furthermore, on these meshes, the refinement fulfills the advantages of newest vertex bisection for n-simplicial adaption.

Chapter 6

Suitability of marked bisection for local refinement in n dimensions

In this chapter, we discuss how to guarantee that the n-dimensional marked bisection algorithm proposed in Chapter 3 can be used for local refinement on unstructured conformal meshes. Accordingly, we need to show that the resulting meshes are conformal, the algorithm finishes in a finite number of steps, and that the meshes asymptotically feature locality. Furthermore, we need to prove a fair bound on the number of generated similarity classes.

The rest of the chapter is organized as follows. First, we illustrate through examples how conformity and reflectivity arise. Second, we discuss how we can guarantee conformity, finiteness, and locality. Finally, we estimate the number of generated similarity classes.

6.1 Marked bisection on a triangle

To illustrate the multi-stage bisection and how the conformity and reflectivity arise, we perform two uniform refinements to a single triangle. First, we mark it with the co-dimensional marking process. Then, we perform two uniform refinements with our marked bisection algorithm. Finally, we check that the resulting mesh is conformal and reflected.

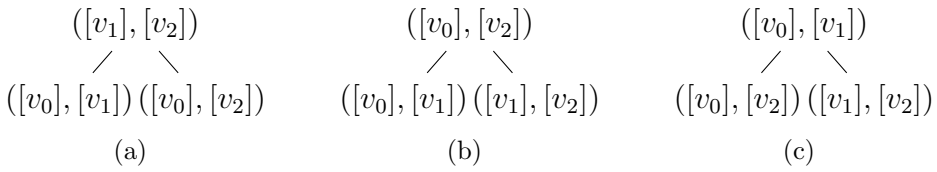


Figure 6.1: Bisection trees of a triangle $\sigma = ([v_0], [v_1], [v_2])$ where the consistent bisection edges is: (a) $e = ([v_1], [v_2])$, (b) $e = ([v_0], [v_2])$, and (c) $e = ([v_0], [v_1])$.

6.1.1 Co-dimensional marks on a triangle

Let $\sigma = ([v_0], [v_1], [v_2])$ be a triangle such that its consistent bisection edge is $e = ([v_1], [v_2])$, and consider the application of the co-dimensional marking process, Algorithm 3.2, to σ. Since e is the consistent bisection edge of σ, we compute the opposite faces to the vertices of e. Thus, we obtain the faces $\kappa_1 = ([v_0], [v_1])$ and $\kappa_2 = ([v_0], [v_2])$.

Then, we apply recursively the co-dimensional marking process to κ_1 and κ_2. Since the dimension of κ_1 is equal to one, the marking process returns as bisection tree t_1 the binary tree composed of the edge $([v_0], [v_1])$, the only edge of κ_1. Analogously, the bisection tree t_2 of κ_2 is the binary tree composed of the edge $([v_0], [v_2])$. Finally, the bisection tree of σ is detailed in Figure 6.1(a).

There are two more cases, depending if the consistent bisection edge of σ is $([v_0], [v_2])$ or $([v_0], [v_1])$. For those cases, the bisection trees generated by the co-dimensional marking process are illustrated in Figure 6.1(b) and 6.1(c), respectively.

Remark 6.1 (Three possible bisection trees for triangles)**.** There are only three possible bisection trees for the 2-dimensional case, depicted in Figure 6.1, because there are only three possible consistent bisection edges of level zero. After choosing the consistent bisection edge of level zero, the rest of the bisection tree is already determined.

6.1.2 Two uniform refinements on a triangle

We refine uniformly two times, with Algorithm 3.1, the tree-simplex $\tau = (\sigma, \bar{\kappa}, t, l)$, with bisection tree t depicted in Figure 6.1(a), and its descendants. We illustrate in Figure 6.2 the obtained meshes during the refinement process. That is, Figures 6.2(a), 6.2(b), and 6.2(c) correspond to $\mathcal{Q}_0^\sigma$, $\mathcal{Q}_1^\sigma$, and $\mathcal{Q}_2^\sigma$, respectively.

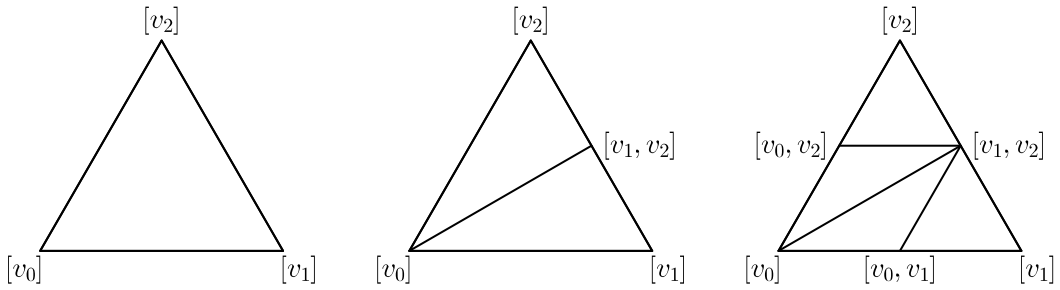


Figure 6.2: Sequence of meshes for two uniform refinements of the triangle σ with the bisection tree depicted in Figure 6.1(a): (a) $\mathcal{Q}_0^\sigma$, (b) $\mathcal{Q}_1^\sigma$, and (c) $\mathcal{Q}_2^\sigma$.

Table 6.1: Triangles after two uniform refinements and corresponding reflected triangles.

Generated triangles	Generated reflected triangles
$\sigma_1 = ([v_0],\, [v_0, v_1],\, [v_1, v_2])$	$\bar{\sigma}_1 = ([v_0],\, [v_0, v_1],\, [v_1, v_2])$
$\sigma_2 = ([v_0, v_1],\, [v_1],\, [v_1, v_2])$	$\bar{\sigma}_2 = ([v_1],\, [v_0, v_1],\, [v_1, v_2])$
$\sigma_3 = ([v_0],\, [v_1, v_2],\, [v_0, v_2])$	$\bar{\sigma}_3 = ([v_0],\, [v_0, v_2],\, [v_1, v_2])$
$\sigma_4 = ([v_0, v_2],\, [v_1, v_2],\, [v_2])$	$\bar{\sigma}_4 = ([v_2],\, [v_0, v_2],\, [v_1, v_2])$

First, we refine uniformly the mesh $\mathcal{Q}_0^\sigma$, generating the mesh $\mathcal{Q}_1^\sigma$, see Figure 6.2(b). This mesh is composed of two tree-simplices τ_1 and τ_2. Each tree-simplex is composed of a triangle $\sigma_1 = ([v_0], [v_1], [v_1, v_2])$ and $\sigma_2 = ([v_0], [v_1, v_2], [v_2])$, respectively. Each tree-simplex has associated the 1-list $\bar{\kappa}_1 = ([v_1, v_2])$ and $\bar{\kappa}_2 = ([v_1, v_2])$, and the bisection tree $t_1 = ([v_0], [v_1])$ and $t_2 = ([v_0], [v_2])$, respectively.

Finally, we refine uniformly $\mathcal{Q}_1^\sigma$, generating the mesh $\mathcal{Q}_2^\sigma$, see Figure 6.2(c), which is composed of four tree-simplices τ_1, τ_2, τ_3, and τ_1. Each tree-simplex τ_i has the associated triangle σ_i, depicted in the first column of Table 6.1.

After generating the four triangles σ_i, the tree-simplex bisection generates the following sorted 2-list $\bar{\kappa}_i$, defined as:

$$\bar{\kappa}_1 = ([v_0, v_1], [v_1, v_2]),\ \bar{\kappa}_2 = ([v_0, v_1], [v_1, v_2]),$$
$$\bar{\kappa}_3 = ([v_0, v_2], [v_1, v_2]),\ \bar{\kappa}_4 = ([v_0, v_2], [v_1, v_2]).$$

Then, we apply cast to Maubach, Algorithm 3.7, and convert all the tree-simplices of $\mathcal{Q}_2^\sigma$ into Maubach triangles $\mu = (\bar{\sigma}, d, l)$. We add in front of each $\bar{\kappa}_i$ the a vertex of the corresponding bisection edge and then, we map those four sorted 3-list into the triangles $\bar{\sigma}_i$, that are depicted in the second column of Table 6.1.

Table 6.2: Mesh inner faces of generated triangles and the reflected triangles.

Inner Faces	Inner reflected faces
$\kappa_1 = \sigma_1 \cap \sigma_2 = \{[v_0, v_1], [v_1, v_2]\}$	$\bar{\kappa}_1 = \bar{\sigma}_1 \cap \bar{\sigma}_2 = ([v_0, v_1], [v_1, v_2])$
$\kappa_2 = \sigma_3 \cap \sigma_4 = \{[v_0, v_2], [v_1, v_2]\}$	$\bar{\kappa}_2 = \bar{\sigma}_3 \cap \bar{\sigma}_4 = ([v_0, v_2], [v_1, v_2])$
$\kappa_3 = \sigma_2 \cap \sigma_3 = \{[v_0], [v_1, v_2]\}$	$\bar{\kappa}_3 = \bar{\sigma}_2 \cap \bar{\sigma}_3 = ([v_0], [v_1, v_2])$

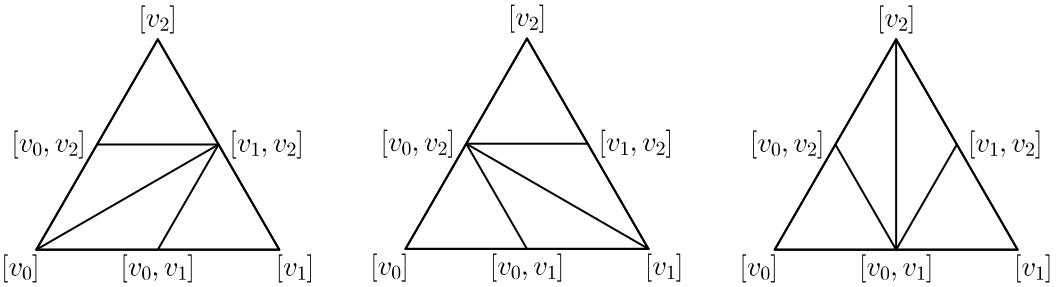


Figure 6.3: The triangular meshes (a), (b) and (c) after two uniform refinements using the balanced bisection trees of Figure 6.1(a), Figure 6.1(b) and Figure 6.1(c), respectively.

6.1.3 Triangular conformity after two refinements

We check that $\mathcal{Q}_2^\sigma$ is conformal and composed of four triangles for the bisection tree of Figure 6.1(a). In the first column of Table 6.2, we detail all the inner faces of $\mathcal{Q}_2^\sigma$. We see that the mesh $\mathcal{Q}_2^\sigma$ is conformal since all the inner edge faces are shared only by two triangles.

Remark 6.2 (Sort preserves conformity)**.** Recall that $\bar{\sigma}_i$ and σ_i have the same vertices. Thus, the mesh $\mathcal{Q}_2^\sigma$ maintains its conformity due the fact that we only sorted the vertices of its triangles.

For the two remaining cases, corresponding to the bisection trees of Figures 6.1(b) and 6.1(c), the obtained meshes $\mathcal{Q}_2^\sigma$ are illustrated in Figures 6.3(b) and 6.3(c), respectively. In all the cases, it is straightforward to check that the obtained triangular meshes are conformal, and composed of four triangles.

6.1.4 Triangular reflectivity after two refinements

We check that $\mathcal{Q}_2^\sigma$ is a reflected mesh. To this end, we check that all the Maubach triangles have the same tag, and that any pair of neighboring triangles share an edge

face that is sorted in the same manner. By construction of the second stage, see Line 6 of Algorithm 3.6, all the Maubach triangles have the same tag $d = 2$. Then, we have to check that any pair of neighboring triangles share an edge face that is sorted in the same manner. The triangles $\bar{\sigma}_i$ of $\mathcal{Q}_2^\sigma$ corresponds to the second column of Table 6.1, and they define a reflected neighbors configuration since the shared edge faces are sorted in the same manner, see second column of Table 6.2. Thus, the mesh $\mathcal{Q}_2^\sigma$ is reflected.

6.2 Marked bisection on a tetrahedron

As in Section 6.1, we proceed in the same manner for a single tetrahedron. First, we mark it with the co-dimensional marking process. Then, we perform three uniform refinements with our marked bisection algorithm. Finally, we check that the resulting mesh is conformal and reflected.

6.2.1 Co-dimensional marks on a tetrahedron

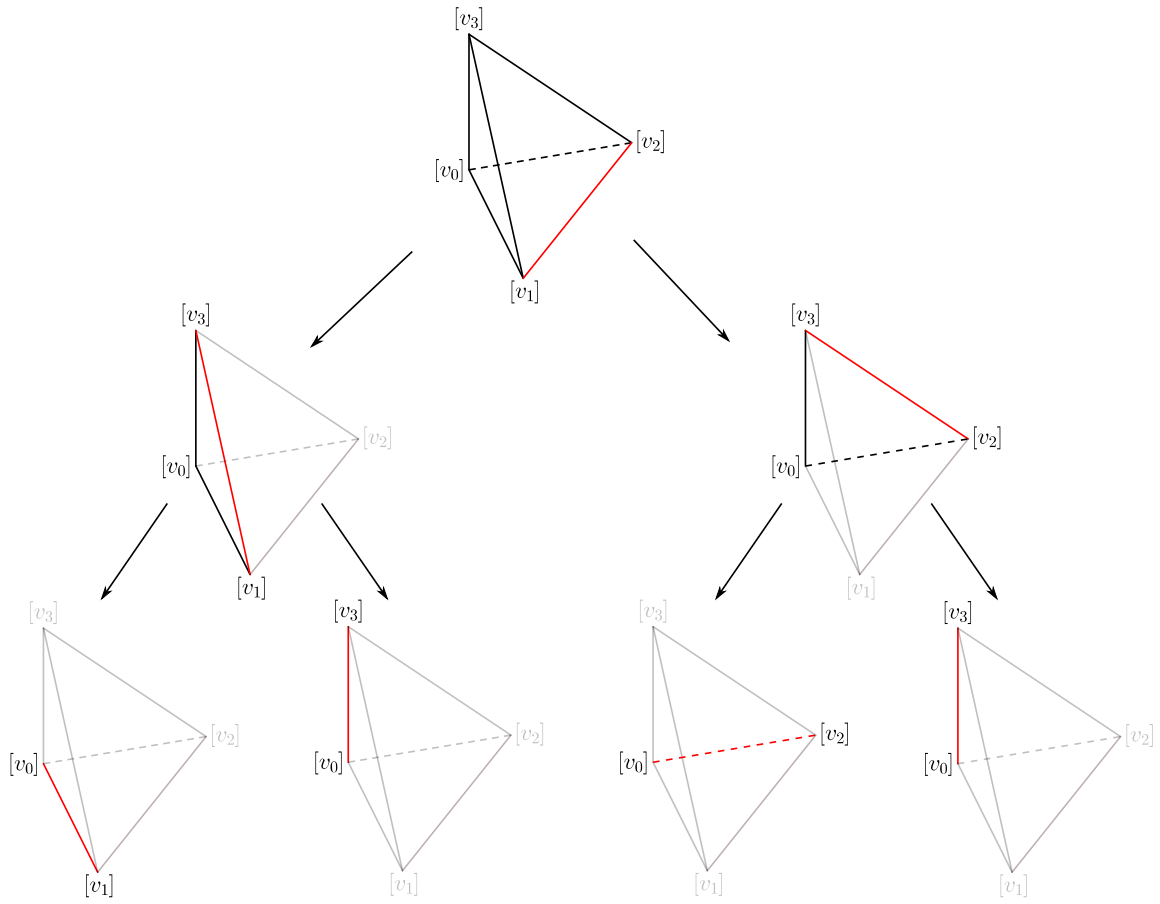
Let $\sigma = ([v_0], [v_1], [v_2], [v_3])$ be a tetrahedron and assume that its edges are ordered in such a manner that

$$e_1 = ([v_1], [v_2]),\ e_2 = ([v_1], [v_3]),\ e_3 = ([v_2], [v_3]),$$
$$e_4 = ([v_0], [v_1]),\ e_5 = ([v_0], [v_2]),\ e_6 = ([v_0], [v_3]),$$

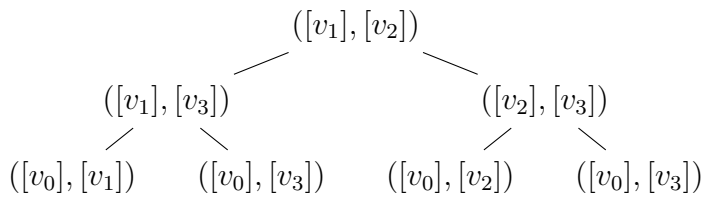
where $e_1 < \ldots < e_6$. Consider the application of the co-dimensional marking process, Algorithm 3.2, to σ.

In Figure 6.4, we show the steps of the co-dimensional marking process and the obtained bisection tree. Figure 6.4(a) shows all the steps that the co-dimensional process makes during the whole process. It has three rows organized by the dimension of the marked entity. Note that the marking process is a co-dimensional algorithm that starts with a tetrahedron and ends with an edge. Thus, the first row is composed of a tetrahedron, the second row is composed of two triangles, and the third row is composed of four edges. Figure 6.4(b) shows the generated bisection tree.

In Figure 6.4(a), we plot the initial tetrahedron in each step and color its vertices and edges to illustrate the current state of the algorithm. We color the vertices in black and gray, depending if they define or not the sub-simplex to be marked. For the edges, we use the same colors as in the vertices. The black edges are the existing



(a)



(b)

Figure 6.4: Each step of the marking process applied to the tetrahedron σ: (a) illustrates each sub-simplex of the marking process and its corresponding consistent bisection edge; and (b) is the obtained bisection tree of σ.

edges of the sub-simplex to be marked, and the gray edges are the edges that are not involved in the marking process. In addition, we colored in red the consistent bisection edge of each sub-simplex.

First, we obtain the consistent bisection edge of σ, which is $e_1 = ([v_1], [v_2])$. It

corresponds to the red edge of the first row of Figure 6.4(a). Since the dimension of the simplex is larger than 1, we obtain the opposite faces to the vertices of e_1, and those faces are the triangles $\kappa_1 = ([v_0], [v_1], [v_3])$ and $\kappa_2 = ([v_1], [v_2], [v_3])$. Those triangles are the left and right triangle of the second row of Figure 6.4(a), respectively. After that, we apply the marking process to κ_1, and then to κ_2.

The consistent bisection edge of κ_1 is $e_2 = ([v_1], [v_3])$, colored in red the left triangle of Figure 6.4(a). Since the dimension of κ_1 is larger than 1, we obtain the opposite faces to the vertices of e_2 inside the triangle κ_1. Those opposite faces are the edges $e_4 = ([v_0], [v_1])$ and $e_6 = ([v_0], [v_3])$, respectively. Again, we apply the marking process to e_4 and e_6. Since the dimension of e_4 is 1, the algorithm returns a tree that only contains a root node with the edge e_4. For the edge e_6, the algorithm proceeds in the same manner, and we obtain a tree with only the node that contains the edge e_6. Those edges correspond to first and second red edges of the third row of Figure 6.4(a). Then, the algorithm builds the bisection tree of the face κ_1. The root node is the bisection edge of κ_1, the edge e_2, and the left and right branches are the bisection trees of the edges e_4 and e_6, respectively.

Next, the algorithm computes analogously the bisection tree of the triangle κ_2. The consistent bisection edge of κ_2 is $e_3 = ([v_2], [v_3])$, the red edge of the right triangle of the second row of Figure 6.4(a). The opposite faces to the vertices of e_3 inside the triangle κ_2 are the edges $e_5 = ([v_0], [v_2])$ and $e_6 = ([v_0], [v_3])$, respectively. Those edges correspond to third and fourth red edges of the third row in Figure 6.4(a). Thus, we apply the marking process to the edges e_5 and e_6. Since the dimension of e_5 and e_6 is 1, the algorithm returns a leaf for each edge such that the root is the corresponding edge. After that, the algorithm returns the bisection tree t_2 of κ_2 with root e_3, and left and right branches corresponding to the bisection trees of e_5 and e_6, respectively.

Finally, the algorithm returns the bisection tree t of σ with root e_1, and left and right branches the bisection trees t_1 and t_2 respectively. Figure 6.4(b) shows the bisection tree t generated by the marking process after marking the tetrahedron σ.

6.2.2 Three uniform refinements on a tetrahedron

We refine uniformly three times, with Algorithm 3.1, the tree-simplex $\tau = (\sigma, \bar{\kappa}, t, l)$, with bisection tree t depicted if Figure 6.4(b), and its descendants. We recall that a tree-simplex of descendant level $l = k$, where $0 \leq k \leq 3$, fulfills that σ is composed

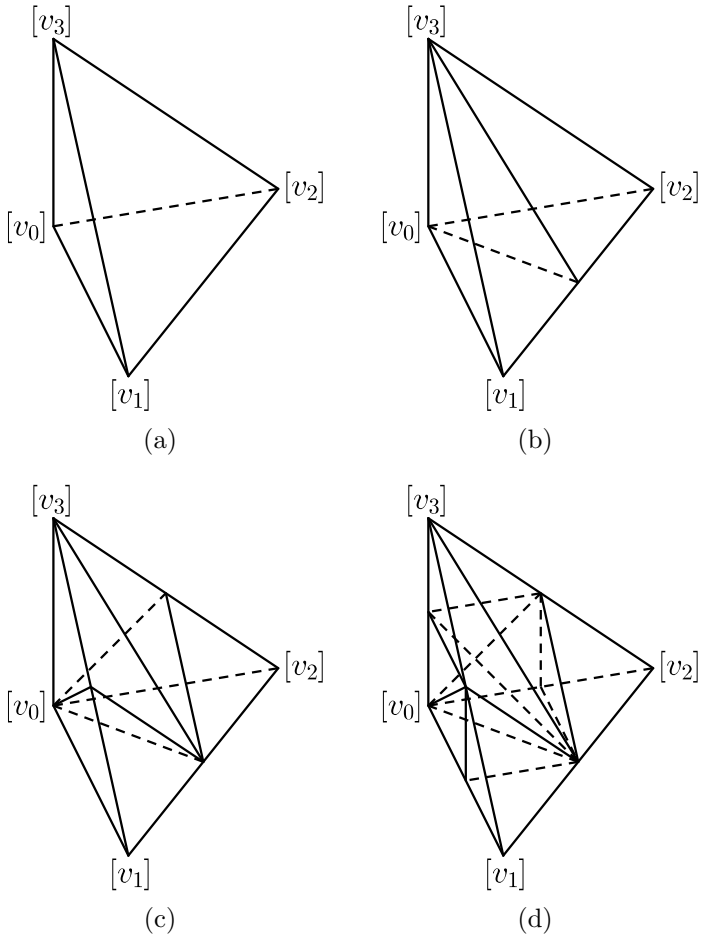


Figure 6.5: Obtained meshes after uniform marked-simplex bisection refinements: (a) $\mathcal{Q}_0^\sigma$, (b) $\mathcal{Q}_1^\sigma$, (c) $\mathcal{Q}_2^\sigma$, and (d) $\mathcal{Q}_3^\sigma$.

of k new vertices and $4-k$ original vertices, a bisection tree t of height $3-k$, and a k-sorted list $\bar{\kappa}$. We illustrate in Figure 6.5 the obtained meshes during the refinement process. That is, Figures 6.5(a), 6.5(b), 6.5(c), and 6.5(d) correspond to $\mathcal{Q}_0^\sigma$, $\mathcal{Q}_1^\sigma$, $\mathcal{Q}_2^\sigma$, and $\mathcal{Q}_3^\sigma$, respectively.

Remark 6.3 (Tree-simplex of descendant level $l = k$)**.** Let σ be a simplex marked with the co-dimensional marking process. Let $\tau = (\sigma, \bar{\kappa}, t, l)$ be the resultant tree-simplex of the marking process. Consider $0 \leq k \leq n$ uniform refinements with marked bisection of τ and its descendants, generating the mesh $\mathcal{Q}_k^\sigma$. Then, the tree-simplices $\tau_i = (\sigma_i, \bar{\kappa}_i, t_i, l_i)$ of $\mathcal{Q}_k^\sigma$ hold that σ_i is composed of k new vertices and $n+1-k$ original vertices, t_i is a bisection tree of height $n-k$, $\bar{\kappa}_i$ is a k-sorted list, and $l_i = k$.

Consider the first uniform marked bisection refinement of $\mathcal{Q}_0^\sigma$. We generate the mesh $\mathcal{Q}_1^\sigma$, see Figure 6.5(b), which is composed of two tree-simplices τ_1 and τ_2. Each tree-simplex is composed of a tetrahedron $\sigma_1 = ([v_0], [v_1], [v_1, v_2], [v_3])$ and $\sigma_2 = ([v_0], [v_1, v_2], [v_2], [v_3])$, respectively. Those tetrahedra share the inner triangular face $\kappa_1 = \{[v_0], [v_1, v_2], [v_3]\}$. Each tree-simplex has associated the 1-list $\bar{\kappa}_1 = ([v_1, v_2])$ and $\bar{\kappa}_2 = ([v_1, v_2])$, and the bisection tree t_1 and t_2, where t_1 and t_2 are

$$\begin{array}{ccc} ([v_1],[v_3]) & & ([v_2],[v_3]) \\ \diagup \quad \diagdown & & \diagup \quad \diagdown \\ ([v_0],[v_1])\,([v_0],[v_3]) & & ([v_0],[v_2])\,([v_0],[v_3]) \end{array}$$

respectively.

Again, refining uniformly $\mathcal{Q}_1^\sigma$ we obtain the mesh $\mathcal{Q}_2^\sigma$, see Figure 6.5(c). This mesh is composed of four tree-simplices τ_1, τ_2, τ_3, and τ_4. Each tree-simplex has the associated tetrahedron

$$\begin{aligned} \sigma_1 &= ([v_0],\, [v_1],\, [v_1, v_2],\, [v_1, v_3]),\ \sigma_2 = ([v_0],\, [v_1, v_3],\, [v_1, v_2],\, [v_3]), \\ \sigma_3 &= ([v_0],\, [v_1, v_2],\, [v_2],\, [v_2, v_3]),\ \sigma_4 = ([v_0],\, [v_1, v_2],\, [v_2, v_3],\, [v_3]). \end{aligned}$$

The bisection creates two new inner triangular faces $\kappa_1 = \{[v_0], [v_1, v_2], [v_1, v_3]\}$ and $\kappa_2 = \{[v_0], [v_1, v_2], [v_2, v_3]\}$. Each tree-simplex has associated the 2-list

$$\begin{aligned} \bar{\kappa}_1 &= ([v_1, v_3],\, [v_1, v_2]),\ \bar{\kappa}_2 = ([v_1, v_3],\, [v_1, v_2]), \\ \bar{\kappa}_3 &= ([v_2, v_3],\, [v_1, v_2]),\ \bar{\kappa}_4 = ([v_2, v_3],\, [v_1, v_2]), \end{aligned}$$

and the bisection tree $t_1 = ([v_0], [v_1])$, $t_2 = ([v_0], [v_3])$, $t_3 = ([v_0], [v_2])$, and $t_4 = ([v_0], [v_3])$, respectively.

Finally, we refine uniformly $\mathcal{Q}_2^\sigma$ and obtain the mesh $\mathcal{Q}_3^\sigma$, see Figure 6.5(d). Recall that the third tree-simplex bisection is performed inside the second stage of our marked bisection scheme. This mesh is composed of eight tree-simplices $\tau_1, \ldots, \tau_8$. Each tree-simplex τ_i has the associated tetrahedron σ_i, depicted in the first column of Table 6.3.

After generating the eight tetrahedra σ_i, the tree-simplex bisection generates the following sorted 3-lists $\bar{\kappa}_i$, defined as:

$$\begin{aligned} \bar{\kappa}_1 &= ([v_0, v_1],\, [v_1, v_3],\, [v_1, v_2]),\ \bar{\kappa}_2 = ([v_0, v_1],\, [v_1, v_3],\, [v_1, v_2]), \\ \bar{\kappa}_3 &= ([v_0, v_3],\, [v_1, v_3],\, [v_1, v_2]),\ \bar{\kappa}_4 = ([v_0, v_3],\, [v_1, v_3],\, [v_1, v_2]), \\ \bar{\kappa}_5 &= ([v_0, v_2],\, [v_2, v_3],\, [v_1, v_2]),\ \bar{\kappa}_6 = ([v_0, v_2],\, [v_2, v_3],\, [v_1, v_2]), \\ \bar{\kappa}_7 &= ([v_0, v_3],\, [v_2, v_3],\, [v_1, v_2]),\ \bar{\kappa}_8 = ([v_0, v_3],\, [v_2, v_3],\, [v_1, v_2]). \end{aligned}$$

Table 6.3: Tetrahedra after three uniform refinements and corresponding reflected tetrahedra.

Generated simplices	Generated reflected simplices
$\sigma_1 = ([v_0], [v_0, v_1], [v_1, v_2], [v_1, v_3])$	$\bar{\sigma}_1 = ([v_0], [v_0, v_1], [v_1, v_3], [v_1, v_2])$
$\sigma_2 = ([v_0, v_1], [v_1], [v_1, v_2], [v_1, v_3])$	$\bar{\sigma}_2 = ([v_1], [v_0, v_1], [v_1, v_3], [v_1, v_2])$
$\sigma_3 = ([v_0], [v_1, v_3], [v_1, v_2], [v_0, v_3])$	$\bar{\sigma}_3 = ([v_0], [v_0, v_3], [v_1, v_3], [v_1, v_2])$
$\sigma_4 = ([v_0, v_3], [v_1, v_3], [v_1, v_2], [v_3])$	$\bar{\sigma}_4 = ([v_3], [v_0, v_3], [v_1, v_3], [v_1, v_2])$
$\sigma_5 = ([v_0], [v_1, v_2], [v_0, v_2], [v_2, v_3])$	$\bar{\sigma}_5 = ([v_0], [v_0, v_2], [v_2, v_3], [v_1, v_2])$
$\sigma_6 = ([v_0, v_2], [v_1, v_2], [v_2], [v_2, v_3])$	$\bar{\sigma}_6 = ([v_2], [v_0, v_2], [v_2, v_3], [v_1, v_2])$
$\sigma_7 = ([v_0], [v_1, v_2], [v_2, v_3], [v_0, v_3])$	$\bar{\sigma}_7 = ([v_0], [v_0, v_3], [v_2, v_3], [v_1, v_2])$
$\sigma_8 = ([v_0, v_3], [v_1, v_2], [v_2, v_3], [v_3])$	$\bar{\sigma}_8 = ([v_3], [v_0, v_3], [v_2, v_3], [v_1, v_2])$

Table 6.4: Mesh inner triangular faces of generated tetrahedra and the reflected tetrahedra.

Inner triangular faces	Reflected inner triangular faces
$\kappa_1 = \sigma_1 \cap \sigma_2 = \{[v_0, v_1], [v_1, v_2], [v_1, v_3]\}$	$\bar{\kappa}_1 = \bar{\sigma}_1 \cap \bar{\sigma}_2 = ([v_0, v_1], [v_1, v_3], [v_1, v_2])$
$\kappa_2 = \sigma_3 \cap \sigma_4 = \{[v_1, v_3]\ [v_1, v_2], [v_0, v_3]\}$	$\bar{\kappa}_2 = \bar{\sigma}_3 \cap \bar{\sigma}_4 = ([v_0, v_3]\ [v_1, v_3], [v_1, v_2])$
$\kappa_3 = \sigma_5 \cap \sigma_6 = \{[v_1, v_2], [v_0, v_2], [v_2, v_3]\}$	$\bar{\kappa}_3 = \bar{\sigma}_5 \cap \bar{\sigma}_6 = ([v_0, v_2], [v_2, v_3], [v_1, v_2])$
$\kappa_4 = \sigma_7 \cap \sigma_8 = \{[v_1, v_2], [v_2, v_3], [v_0, v_3]\}$	$\bar{\kappa}_4 = \bar{\sigma}_7 \cap \bar{\sigma}_8 = ([v_0, v_3], [v_2, v_3], [v_1, v_2])$
$\kappa_5 = \sigma_1 \cap \sigma_3 = \{[v_0], [v_1, v_2], [v_1, v_3]\}$	$\bar{\kappa}_5 = \bar{\sigma}_1 \cap \bar{\sigma}_3 = ([v_0], [v_1, v_3], [v_1, v_2])$
$\kappa_6 = \sigma_5 \cap \sigma_7 = \{[v_0], [v_1, v_2], [v_2, v_3]\}$	$\bar{\kappa}_6 = \bar{\sigma}_5 \cap \bar{\sigma}_7 = ([v_0], [v_2, v_3], [v_1, v_2])$
$\kappa_7 = \sigma_3 \cap \sigma_7 = \{[v_0], [v_1, v_2], [v_0, v_3]\}$	$\bar{\kappa}_7 = \bar{\sigma}_3 \cap \bar{\sigma}_7 = ([v_0], [v_0, v_3], [v_1, v_2])$
$\kappa_8 = \sigma_4 \cap \sigma_8 = \{[v_0, v_3], [v_1, v_2], [v_3]\}$	$\bar{\kappa}_8 = \bar{\sigma}_4 \cap \bar{\sigma}_8 = ([v_3], [v_0, v_3], [v_1, v_2])$

Then, we apply the cast to Maubach, Algorithm 3.7, and convert all the tree-simplices of $\mathcal{Q}_3^\sigma$ into Maubach simplices $\mu = (\bar{\sigma}, d, l)$. We add in front of each $\bar{\kappa}_i$ the a vertex of the corresponding bisection edge and then, we map those eight sorted 4-list into the tetrahedra $\bar{\sigma}_i$, that are depicted in the second column of Table 6.3.

6.2.3 Tetrahedral conformity after three refinements

We want to check that, for the presented tetrahedron σ marked with the co-dimensional marking process, the obtained mesh $\mathcal{Q}_3^\sigma$ is conformal, and that all the edges and only the edges of σ have been bisected. That is, the bisection tree t generated by the co-dimensional marking process is balanced.

We see that all the inner faces are shared by two tetrahedra, see first column of Table 6.4. Therefore, the mesh $\mathcal{Q}_3^\sigma$ is conformal. Moreover, since t contains all the edges of σ and we used the entire tree, all the edges and only the edges of σ have

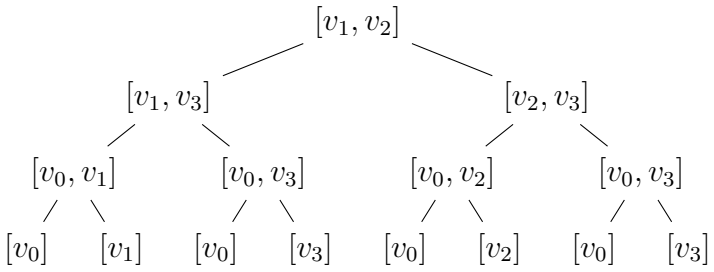


Figure 6.6: Complete vertex tree $\bar{t}$ corresponding to the bisection tree of Figure 6.4(b).

been bisected.

6.2.4 Tetrahedral reflectivity after three refinements

Next, we check that $\mathcal{Q}_3^\sigma$ is a reflected mesh. To this end, we check that all the Maubach tetrahedra have the same tag, and that any pair of neighboring tetrahedra share a triangular face that is sorted in the same manner. By construction of the second stage, see Line 6 of Algorithm 3.6, all the Maubach tetrahedra have the same tag $d = 3$. Then, we have to check that any pair of neighboring tetrahedra share a triangular face that is sorted in the same manner. The tetrahedra $\bar{\sigma}_i$ of $\mathcal{Q}_3^\sigma$ correspond to the second column of Table 6.3, and they define a reflected neighbors configuration since the shared triangular faces are sorted in the same manner, see second column of Table 6.4. Thus, as we wanted to see, the mesh $\mathcal{Q}_3^\sigma$ is reflected.

Remark 6.4 (Reflectivity induced by the complete vertex tree)**.** The idea behind the reflectivity of the mesh $\mathcal{Q}_3^\sigma$ is that the order of the vertices of $\bar{\sigma}$ is induced by the complete vertex tree $\bar{t}$ associated to the bisection tree t.

From Section 2.2.3, we know that each branch of $\bar{t}$, depicted in Figure 6.6, defines a tetrahedron of $\mathcal{Q}_3^\sigma$. Let $\bar{\sigma}$ be a tetrahedron of $\mathcal{Q}_3^\sigma$, which has the form

$$\bar{\sigma} = ([v_{i,2}], [v_{1,2}, v_{2,2}], [v_{1,1}, v_{2,1}], [v_{1,0}, v_{2,0}]),$$

where $i \in \{1, 2\}$, by construction of the Algorithm 3.7, see Lines 5–6. The order of the vertices of $\bar{\sigma}$ correspond to sorting the proper branch of $\bar{t}$ from the leaf to the root. Thus, if we sort each branch of $\bar{t}$ from the leaf to the root, we have sorted all the tetrahedra in a manner that all the shared entities between tetrahedra have the same order. Particularly, the inner triangular faces have the same order in adjacent

tetrahedra and therefore, the mesh is reflected. This reasoning will be used to prove that $\mathcal{Q}_n$ is a reflected mesh.

6.3 Conformity and reflectivity after n uniform refinements

We discuss next how we can guarantee conformity, finiteness, and asymptotic locality. To this end, since our method switches independently to the newest vertex bisection at the n-th bisection of each simplex, it is sufficient to guarantee that after n uniform refinements of the whole mesh, the resulting mesh fulfills sufficient conditions to use the newest vertex bisection. Hence, it will inherit from the newest vertex bisection the subsequent conformity, finiteness, and asymptotic locality. To use the newest vertex bisection, it is sufficient to check that the mesh is conformal and reflected, see Remark 2.6. Accordingly, to guarantee that our multi-stage bisection can be used for local refinement, we propose to prove that after n uniform refinements the meshes are conformal and reflected. Although this is a proof for the worst case, it does not imply that in practice the method needs to uniformly refine n times the initial mesh. This reasoning is analogous to the proof that three-dimensional marked bisection terminates on a conformal mesh when starting on conformingly-marked meshes, see Theorem 3.1 in Arnold et al. (2000). Specifically, the authors consider multiple iterations of local refinement with marked bisection for a conformingly-marked mesh. They prove that in the worst case those local refinements behave as a succession of uniform refinements of the whole mesh.

Following, for n uniform refinements, we guarantee that the mesh is reflected, and we outline how to prove that the mesh is conformal.

6.3.1 Simplicial reflectivity after n uniform refinements

Following, we prove that after uniformly refining n times a conformal simplicial mesh $\mathcal{Q}_0 = \mathcal{T}_0$, the obtained mesh $\mathcal{Q}_n$ is a reflected mesh. The idea is to prove that $\mathcal{Q}_n$, after using cast to Maubach, is a sorted simplicial mesh such that all its simplices have tag $d = n$. Then, by Theorem 5.1, we can ensure that the mesh $\mathcal{Q}_n$ is reflected.

As we see in Section 2.2.3, each branch of a complete vertex tree $\bar{t}$ defines a simplex of $\mathcal{Q}_n$. Thus, the order of the vertices of each simplex of $\mathcal{Q}_n$ corresponds

to traversing from the leaf to the root the corresponding branch of $\bar{t}$. We define an order of vertices, $\prec$, that is compatible with the order of vertices of each simplex in the following manner

(1) $[v_i] \prec [v_j]$ if $v_i < v_j$.

(2) Let $\mathbf{v}_1 = [v_{i_1}, v_{j_1}]$ and $\mathbf{v}_2 = [v_{i_2}, v_{j_2}]$ be two vertices of length 2, that correspond to the midpoint of the edges $e_1 = ([v_{i_1}], [v_{j_1}])$ and $e_2 = ([v_{i_2}], [v_{j_2}])$. We say that $\mathbf{v}_1 \prec \mathbf{v}_2$ if $e_2 < e_1$, where $<$ is the strict total order of the mesh edges.

(3) Let $\mathbf{v}_1$ and $\mathbf{v}_2$ be two vertices of length k_1 and k_2, respectively. We say that $\mathbf{v}_1 \prec \mathbf{v}_2$ if $k_1 < k_2$.

Lemma 6.1. *The order $\prec$ is strict and total.*

Proof. Is straightforward to check that $\prec$ is irreflexive, asymmetric, and transitive since we have used a strict total order for the mesh vertices and the mesh edges of $\mathcal{T}_0$. □

Theorem 6.1 (Sorted mesh after n uniform refinements)**.** *The mesh $\mathcal{Q}_n$ is a sorted simplicial mesh with the order $\prec$.*

Proof. We need to check that the mesh is composed of sorted simplices with the order $\prec$. Let p be the set of vertices corresponding to a branch of $\bar{t}$. Since p is a branch of $\bar{t}$, it has the form

$$p = \{[v_{i,n-1}], [v_{1,n-1}, v_{2,n-1}], \ldots, [v_{1,0}, v_{2,0}]\},$$

where $i \in \{1, 2\}$, see Lines 5–6 of Algorithm 3.7. Since each branch of $\bar{t}$ correspond to a simplex $\bar{\sigma}$ of $\mathcal{Q}_n$, we have that p corresponds to the set of vertices of $\bar{\sigma}$. Consider the application of $\prec$ to p, obtaining that

$$[v_{i,n-1}] \prec [v_{1,n-1}, v_{2,n-1}] \prec \ldots \prec [v_{1,0}, v_{2,0}].$$

That is because $[v_{1,l+1}, v_{2,l+1}] \prec [v_{1,l}, v_{2,l}]$ from (2), where $0 \leq l \leq n-2$, and $[v_{i,n-1}] \prec [v_{1,n-1}, v_{2,n-1}]$ from (3). Thus, the simplex

$$\bar{\sigma} = ([v_{i,n-1}], [v_{1,n-1}, v_{2,n-1}], \ldots, [v_{1,0}, v_{2,0}]) = \texttt{simplex}\ p,$$

where `simplex` is defined in Equation 5.4, is a sorted simplex with the strict total order $\prec$. Since this reasoning can be applied to all the simplices of $\mathcal{Q}_n$, we conclude that $\mathcal{Q}_n$ is a sorted mesh with the strict total order $\prec$. □

Corollary 6.1 (Reflectivity after n uniform refinements)**.** $\mathcal{Q}_n$ is a reflected mesh.

Proof. By Theorem 6.1, the mesh $\mathcal{Q}_n$ is a sorted simplicial mesh. Since the marked bisection assigns the same tag $d = n$ to all the simplices of $\mathcal{Q}_n$, by Theorem 5.1 we can ensure that $\mathcal{Q}_n$ is a reflected mesh. □

6.3.2 Conformity after n uniform refinements

To prove the conformity, we split the reasoning into two parts: the external and the internal conformity. For the external conformity, we show that a shared face κ between two simplices is bisected in the same manner from both sides. Then, for the internal conformity, we show that when we refine uniformly n times a simplex σ, the inner faces that appear during the refinement process will be uniquely bisected. We check the conformity through mesh faces because Theorem 2.1 of Stevenson (2008) ensure that is sufficient. Before, we conjecture some results that we will assume as true in the outline of the conformity proof.

Conjecture 6.1 (Isomorphic bisection trees)**.** *Let $\mathcal{T}$ be a conformal simplicial mesh marked with our co-dimensional marking process. Let $\tau \in \mathcal{T}$ by a k-entity shared by σ_1 and σ_2. Let t_1 and t_2 be the inherited bisection trees τ when τ is marked from σ_1 and σ_2, respectively. Then, t_1 and t_2 are the same up to the order of appearance of the vertices.*

Conjecture 6.2 (Bisection of k-entities)**.** *Let $\mathcal{T}$ be a conformal simplicial mesh marked with our co-dimensional marking process, and let $\tau \in \mathcal{T}$ be a k-entity with bisection tree t. Then, $\mathcal{Q}_k^\tau$ depends only on t and $\mathcal{Q}_k^\tau$ is the same for equivalent bisection trees obtained with different order of vertices.*

Remark 6.5. Conjectures 6.1 and 6.2 ensure that using our co-dimensional marking process, the mesh $\mathcal{Q}_k^\tau$ of a refined k-entity τ is the same from all the simplices that contain it.

Proposition 6.1 (Conformity on external faces)**.** *Let $\mathcal{T}_0$ be a conformal simplicial mesh marked with our co-dimensional marking process, and let σ_1 and σ_2 two neighboring simplices that share a face κ. Then, the mesh $\mathcal{Q}_n$ is conformal through the mesh faces defined by the simplices of the refined mesh $\mathcal{Q}_{n-1}^\kappa$.*

Proof. We want to check that the mesh $\mathcal{Q}_n$ is conformal through the interfaces generated by mesh faces of $\mathcal{T}_0$. To this end, we check that for every pair of neighboring simplices of $\mathcal{T}_0$ that share a face, after n uniform refinements, maintain their conformity through the shared face.

Let $\sigma_1, \sigma_2 \in \mathcal{T}_0$ be two neighboring simplices that share a face $\kappa = \sigma_1 \cap \sigma_2$. By Conjecture 6.1, the bisection tree of κ is the same, up to isomorphism, inherited from σ_1 and σ_2. Therefore, by Conjecture 6.2, the mesh $\mathcal{Q}_{n-1}^{\kappa}$ is the same from $\mathcal{Q}_n^{\sigma_1}$ and $\mathcal{Q}_n^{\sigma_2}$. Thus, the mesh is conformal through $\mathcal{Q}_{n-1}^{\kappa}$. Since we can repeat this for every pair of neighboring simplices of $\mathcal{T}_0$ that share a face, we conclude that $\mathcal{Q}_n$ is conformal. □

Proposition 6.2 (Conformity on internal faces)**.** *Let σ be a simplex marked with our co-dimensional marking process. Then, $\mathcal{Q}_n^{\sigma}$ is conformal.*

Proof. We want to prove that $\mathcal{Q}_n^{\sigma}$ is conformal. To this end, we will use induction. By Section 6.1, we know that $\mathcal{Q}_2^{\sigma}$ is conformal and this is the base case of the induction. Our induction hypothesis is that a $(n-1)$-simplex σ marked with our co-dimensional marking process leads to a conformal mesh $\mathcal{Q}_{n-1}^{\sigma}$ after $n-1$ uniform refinements. In this outline of the proof, we use the tetrahedral case illustrated in Figure 6.7 as support, since the procedure used to prove the 3-dimensional one can be extended to the n-dimensional case.

Let $\sigma = \mathcal{Q}_0^{\sigma}$ be an n-simplex marked with our co-dimensional marking process, see Figure 6.7(a). Consider the bisection of σ using the consistent bisection edge of level zero, e. In this step, we generate the mesh $\mathcal{Q}_1^{\sigma}$ and the new vertex ν, see Figure 6.7(b). This mesh is composed of two simplices σ_1 and σ_2, and is a conformal mesh. Those simplices are composed of the opposite faces κ_1 and κ_2 to the vertices of e, respectively, and ν.

Consider the $(n-1)$-simplicial mesh composed of the opposites faces κ_1 and κ_2, obtained after removing the vertex ν, see Figure 6.7(c). By induction hypothesis, the meshes $\mathcal{Q}_{n-1}^{\kappa_1}$ and $\mathcal{Q}_{n-1}^{\kappa_2}$ are conformal, see Figure 6.7(d).

The faces κ_1 and κ_2 share an $(n-2)$-entity τ, that is the opposite $(n-2)$-entity to the consistent bisection edge of level zero e. By Conjecture 6.1, the bisection trees of τ inherited from κ_1 and κ_2 are isomorphic. Thus, according to Conjecture 6.2, the bisected mesh of τ will be the same from κ_1 and κ_2. Therefore, we can merge the meshes $\mathcal{Q}_{n-1}^{\kappa_1}$ and $\mathcal{Q}_{n-1}^{\kappa_2}$ through the common interface in a conformal manner, see Figure 6.7(e). Finally, we complete the $(n-1)$-simplices by adding

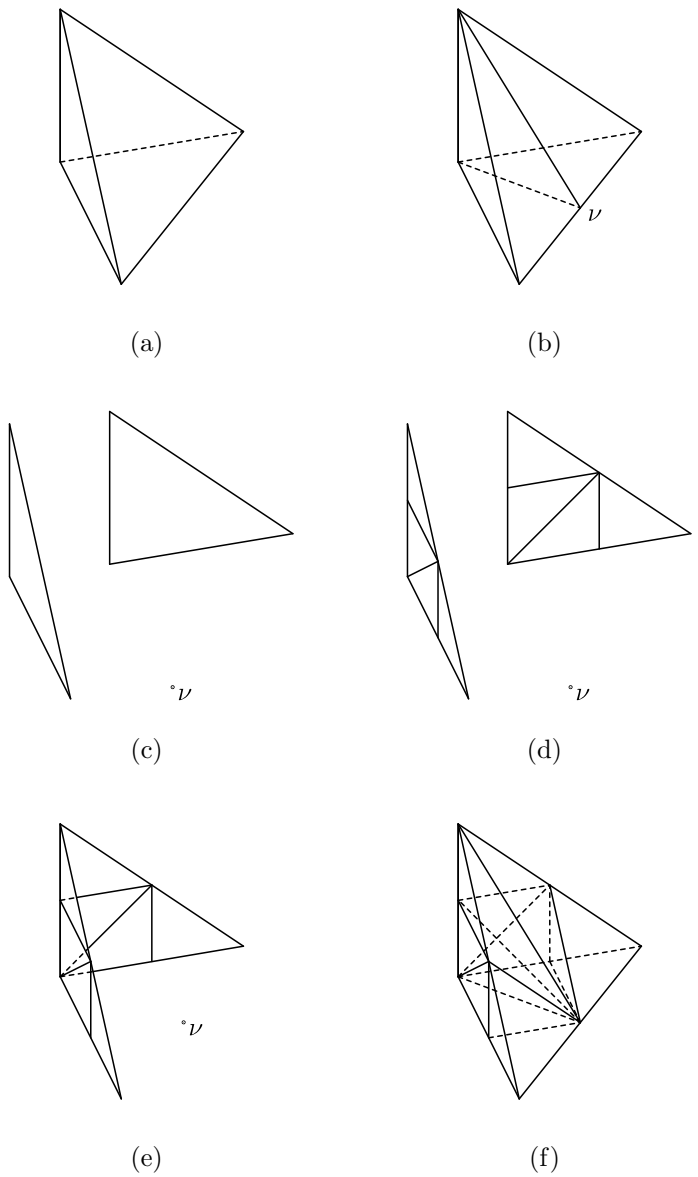


Figure 6.7: Outline of the proof: (a) Given a tetrahedron, and a refinement edge e with midpoint ν, (b) trough bisection we obtain two tetrahedra. Then, (c) we split the two tetrahedra into two triangles that share an edge and the vertex ν. After that, (d) we perform two uniform refinements to the triangles. Thus, since the shared edge is refined in the same manner, (e) we can be merged the triangular meshes trough their shared refined edge. Finally, (f) we connect ν to all the vertices of the merged triangular mesh, generating a conformal tetrahedral mesh composed of 8 tetrahedra.

the vertex ν, see Figure 6.7(f), and generating the n-simplicial mesh $\mathcal{Q}_n^\sigma$. Since this operation maintains the conformity of the mesh, the obtained n-simplicial mesh $\mathcal{Q}_n^\sigma$

is conformal. □

Theorem 6.2 (Bisection conformity condition)**.** *Let $\mathcal{T}_0$ be a conformal simplicial mesh and $\mathcal{T}_1$ be the simplicial mesh obtained with a bisection method. Assume that the mesh $\mathcal{T}_1$ is conformal through the faces that are descendants of the faces of mesh $\mathcal{T}_0$, and conformal through the bisection faces obtained inside the simplices of $\mathcal{T}_0$. Then, the $\mathcal{T}_1$ is conformal.*

Proof. According to Theorem 2.1, we need to check the conformity of the mesh $\mathcal{T}_1$ through the faces. There are two types of inner faces in $\mathcal{T}_1$: the faces that are descendants of faces in $\mathcal{T}_0$; and the faces that are created inside simplices of $\mathcal{T}_0$. In both cases, the hypotheses ensure the conformity of the mesh $\mathcal{T}_1$ through its faces, and therefore the mesh $\mathcal{T}_1$ is conformal. □

Corollary 6.2 (Conformity after n uniform refinements)**.** Let $\mathcal{T}_0$ be a conformal simplicial mesh, and $\mathcal{Q}_n$ be the mesh obtained after n uniform refinements using our marked bisection. Then, the mesh $\mathcal{Q}_n$ is conformal.

Proof. Propositions 6.1 and 6.2 ensure that the mesh $\mathcal{Q}_n$ satisfies the hypotheses of Theorem 6.2. Thus, the mesh $\mathcal{Q}_n$ is conformal. □

6.4 Estimation of the number of similarity classes

Finally, we estimate an upper bound of the number of similarity classes obtained with our marked bisection. We prove that this number is sub-optimal. That is, it is slightly greater than the number of similarity classes obtained using newest vertex bisection.

Lemma 6.2 (Newest vertex bisection for triangular meshes)**.** *The marked bisection is equivalent to Maubach's bisection for 2-simplices.*

Proof. We have seen that the co-dimensional marking process generates three possible bisection trees for 2-simplices. Those bisection trees are

$$\begin{array}{ccc} ([v_1],[v_2]) & ([v_0],[v_2]) & ([v_0],[v_1]) \\ \diagup \quad \diagdown & \diagup \quad \diagdown & \diagup \quad \diagdown \\ ([v_0],[v_1])\,([v_0],[v_2]) & ([v_0],[v_1])\,([v_1],[v_2]) & ([v_0],[v_2])\,([v_1],[v_2]) \end{array}$$

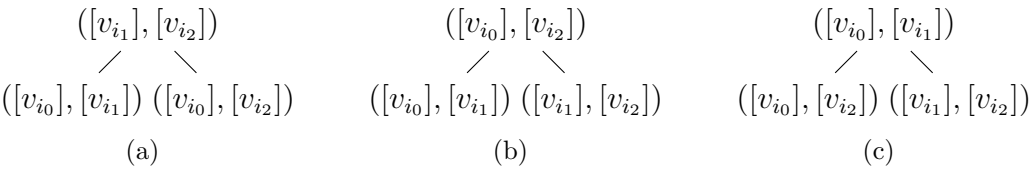


Figure 6.8: The three possible bisection trees t_i corresponding to the consistent bisection edges (a) $([v_{i_1}],[v_{i_2}])$, (b) $([v_{i_0}],[v_{i_2}])$, and (c) $([v_{i_0}],[v_{i_1}])$.

, trees that are equivalent to the bisection trees generated by the tagged triangles $(([v_1],[v_0],[v_2]), d = 2)$, $(([v_0],[v_1],[v_2]), d = 2)$, and $(([v_0],[v_2],[v_1]), d = 2)$, respectively. Therefore, for triangular meshes our marked bisection algorithm is equivalent to Maubach's algorithm. □

Proposition 6.3 (Tagging with $d = 2$ after step $n-2$)**.** *Let σ be a simplex marked with the co-dimensional marking process, and consider the mesh $\mathcal{Q}^{\sigma}_{n-2}$ obtained after $n-2$ uniform refinements with marked bisection. Then, we can map a tree-simplex τ of $\mathcal{Q}^{\sigma}_{n-2}$ to a Maubach simplex μ with descendant level $l = n-2$ and tag $d = 2$.*

Proof. Let σ_0 be a simplex marked with the co-dimensional marking process and $\mathcal{Q}^{\sigma_0}_{n-1}$ be the mesh obtained after $n-2$ uniform marked bisection refinements. Let $\tau \in \mathcal{Q}^{\sigma_0}_{n-2}$ be a tree-simplex of the form $\tau = (\sigma, \bar{\kappa}, t, l = n-2)$. After applying $n-2$ uniform refinements with marked bisection, we know that σ is composed of 3 original vertices and $n-2$ multivertices. That is,

$$\sigma = \{[v_{i_0}],[v_{i_1}],[v_{i_2}],[v_{1,n-3},v_{2,n-3}],\ldots,[v_{1,0},v_{2,0}]\}.$$

Since the bisection tree t of σ is composed of vertices $[v_{i_0}]$, $[v_{i_1}]$, and $[v_{i_2}]$, that define a triangle, its bisection tree is equivalent to the bisection tree of a triangle. By Lemma 6.2, t is equivalent to the bisection tree of a tagged triangle. Thus, t is one of the bisection trees depicted in Figure 6.8.

If we map the tree-simplex τ corresponding to the simplex σ to a Maubach simplex $\mu = (\bar{\sigma}, l = n-2, d = 2)$, we have that there are three possible simplices $\bar{\sigma}$, illustrated in Equation (6.1):

$$\bar{\sigma} = ([v_{i_1}],[v_{i_0}],[v_{i_2}],[v_{1,n-3},v_{2,n-3}],\ldots,[v_{1,0},v_{2,0}]), \tag{6.1a}$$

$$\bar{\sigma} = ([v_{i_0}],[v_{i_1}],[v_{i_2}],[v_{1,n-3},v_{2,n-3}],\ldots,[v_{1,0},v_{2,0}]), \tag{6.1b}$$

$$\bar{\sigma} = ([v_{i_0}],[v_{i_2}],[v_{i_1}],[v_{1,n-3},v_{2,n-3}],\ldots,[v_{1,0},v_{2,0}]). \tag{6.1c}$$

We recall that we sorted the vertices $[v_{i_0}]$, $[v_{i_1}]$, and $[v_{i_2}]$ according to the tagged triangles of the proof of Lemma 6.2. Thus, we have that the simplices of Equations (6.1a)–(6.1c) correspond to the bisection trees depicted in Figures 6.8(a)–6.8(c).

Then, it only remains to check that if we apply two uniform tagged-bisection steps to $\bar{\sigma}_i$, the obtained Maubach simplices are the same than the Maubach simplex obtained after n uniform refinements with marked bisection.

Let $\mu = (\bar{\sigma}, l = n-2, d = d = 2)$ be a Maubach simplex, where $\bar{\sigma}$ is defined in Equation (6.1a). Performing the first Maubach's bisection step, we obtain two children $\mu_1 = (\bar{\sigma}_1, l = n-1, d = 1)$ and $\mu_2 = (\bar{\sigma}_2, l = n-1, d = 1)$, where

$$\begin{aligned}\bar{\sigma}_1 &= ([v_{i_1}], [v_{i_0}], [v_{i_1}, v_{i_2}], [v_{1,n-3}, v_{2,n-3}], \ldots, [v_{1,0}, v_{2,0}]),\\ \bar{\sigma}_2 &= ([v_{i_0}], [v_{i_2}], [v_{i_1}, v_{i_2}], [v_{1,n-3}, v_{2,n-3}], \ldots, [v_{1,0}, v_{2,0}]).\end{aligned}$$

Since the edge $([v_{i_1}], [v_{i_2}])$ is the consistent bisection edge of level $l = n-2$, we have that $[v_{i_1}, v_{i_2}] = [v_{1,n-2}, v_{2,n-2}]$. Thus, if we substitute it on $\bar{\sigma}_1$ and $\bar{\sigma}_2$, we have that

$$\begin{aligned}\bar{\sigma}_1 &= ([v_{i_1}], [v_{i_0}], [v_{1,n-2}, v_{2,n-2}], [v_{1,n-3}, v_{2,n-3}], \ldots, [v_{1,0}, v_{2,0}]),\\ \bar{\sigma}_2 &= ([v_{i_0}], [v_{i_2}], [v_{1,n-2}, v_{2,n-2}], [v_{1,n-3}, v_{2,n-3}], \ldots, [v_{1,0}, v_{2,0}]).\end{aligned}$$

Again, we perform a uniform tagged-bisection to μ_1 and μ_2 obtaining four Maubach simplices $\mu_1 = (\bar{\sigma}_1, l = n, d = n)$, $\mu_2 = (\bar{\sigma}_2, l = n, d = n)$, $\mu_3 = (\bar{\sigma}_3, l = n, d = n)$, and $\mu_4 = (\bar{\sigma}_4, l = n, d = n)$, where

$$\begin{aligned}\bar{\sigma}_1 &= ([v_{i_1}], [v_{i_0}, v_{i_1}], [v_{1,n-2}, v_{2,n-2}], \ldots, [v_{1,0}, v_{2,0}]),\\ \bar{\sigma}_2 &= ([v_{i_0}], [v_{i_0}, v_{i_1}], [v_{1,n-2}, v_{2,n-2}], \ldots, [v_{1,0}, v_{2,0}]),\\ \bar{\sigma}_3 &= ([v_{i_0}], [v_{i_0}, v_{i_2}], [v_{1,n-2}, v_{2,n-2}], \ldots, [v_{1,0}, v_{2,0}]),\\ \bar{\sigma}_4 &= ([v_{i_2}], [v_{i_0}, v_{i_2}], [v_{1,n-2}, v_{2,n-2}], \ldots, [v_{1,0}, v_{2,0}]).\end{aligned}$$

Analogously, the consistent bisection edges of level $l = n-1$ are $([v_{i_0}], [v_{i_1}])$ and $([v_{i_0}], [v_{i_2}])$. Therefore, we have $[v_{i_0}, v_{i_1}] = [v_{1,n-1}, v_{2,n-1}]$ for $\bar{\sigma}_1$ and $\bar{\sigma}_2$, and $[v_{i_0}, v_{i_2}] = [v_{1,n-1}, v_{2,n-1}]$ for $\bar{\sigma}_3$ and $\bar{\sigma}_4$. Finally, we have that $[v_{i_0}] = [v_{1,n-1}]$ and $[v_{i_1}] = [v_{2,n-1}]$ for $\bar{\sigma}_1$ and $\bar{\sigma}_2$, and $[v_{i_0}] = [v_{1,n-1}]$ and $[v_{i_2}] = [v_{2,n-1}]$ for $\bar{\sigma}_3$ and $\bar{\sigma}_4$. Thus, all the simplices can be expressed as

$$\begin{aligned}\bar{\sigma}_1 &= ([v_{1,n-1}], [v_{1,n-1}, v_{2,n-1}], [v_{1,n-2}, v_{2,n-2}], \ldots, [v_{1,0}, v_{2,0}]),\\ \bar{\sigma}_2 &= ([v_{2,n-1}], [v_{1,n-1}, v_{2,n-1}], [v_{1,n-2}, v_{2,n-2}], \ldots, [v_{1,0}, v_{2,0}]),\\ \bar{\sigma}_3 &= ([v_{1,n-1}], [v_{1,n-1}, v_{2,n-1}], [v_{1,n-2}, v_{2,n-2}], \ldots, [v_{1,0}, v_{2,0}]),\\ \bar{\sigma}_4 &= ([v_{2,n-1}], [v_{1,n-1}, v_{2,n-1}], [v_{1,n-2}, v_{2,n-2}], \ldots, [v_{1,0}, v_{2,0}]).\end{aligned}$$

The obtained simplices are the same that the simplices obtained after n uniform refinements with marked bisection. That is, $\bar{\sigma}_1$ and $\bar{\sigma}_3$ are equal to the simplex of Line 5 of Algorithm 3.7, and $\bar{\sigma}_2$ and $\bar{\sigma}_4$ are equal to the simplex of Line 5 of Algorithm 3.7.

This procedure can be applied to the remaining two cases where the initial $\bar{\sigma}$ is defined by Equations (6.1b) and (6.1c). Thus, we have proved that we can map a tree-simplex $\tau \in \mathcal{Q}_{n-2}^{\sigma_0}$ to a Maubach simplex μ with descendant level $l = n-2$ and tag $d = 2$. □

Now, we can state Theorem 6.3 where we give an upper bound S_n over the similarity classes generated by our marked bisection algorithm.

Theorem 6.3 (Number of similarity classes of marked bisection)**.** *Let σ be a simplex marked with the co-dimensional marking process. Assume that from iteration k, the bisection process is equivalent to Maubach's bisection. Then, the number of similarity classes generated by our marked bisection method is at most*

$$S_n = (2^k - 1) + 2^k M_n = (2^k - 1) + 2^k n n! 2^{n-2}.$$

Proof. Let σ be a simplex marked with the co-dimensional marking process. Consider k uniform marked bisection refinements such that further refinements of marked bisection are equivalent to Maubach's bisection. By Equation (5.8), the number of similarity classes generated from iteration 0 to $k-1$ is $2^k - 1$. Since the number of simplices of $\mathcal{Q}_k^\sigma$ is 2^k, the number of simplices generated using Maubach's algorithm is $2^k M_n$, where M_n is a bound of similarity classes of Maubach's algorithm, see Theorem 4.5 of Arnold et al. (2000). Finally, summing the two values we obtain that the number of similarity classes is at most $S_n = (2^k - 1) + 2^k M_n$, as we wanted to see. □

Corollary 6.3. In our marked bisection, k is at most $n-2$, and the number of similarity classes is at most

$$S_n = (2^{n-2} - 1) + 2^{n-2} M_n = (2^{n-2} - 1) + 2^{n-2} n n! 2^{n-2}.$$

Proof. By Proposition 6.3, at iteration $n-2$ we can map all the simplices of $\mathcal{Q}_{n-2}^\sigma$ to Maubach simplices with tag descendant level $l = n-2$ and tag $d = 2$. Therefore, by Theorem 6.3, the number of similarity classes generated by our marked bisection algorithm is at most

$$S_n = (2^{n-2} - 1) + 2^{n-2} M_n.$$

□

6.5 Conclusions

In conclusion, we have seen that the proposed multi-stage marked bisection can be used for local refinement of unstructured conformal meshes. Specifically, after illustrating the cases for two and three dimensions, we have outlined a worst case proof to see that our multi-stage marked bisection is conformal, finite, and asymptotically local. The proof only depends on the proving that our method satisfies Conjectures 6.1 and 6.2. Furthermore, we have derived an estimation of generated similarity classes, an estimation that shows this bound is slightly bigger than the bound for the newest vertex bisection. All these features justify the suitability for local refinement.

Chapter 7

Conclusions and future work

In this thesis, we have demonstrated conformal bisection methods in arbitrary dimensions for local refinement of unstructured conformal simplicial meshes. To this end, we have fulfilled the following objectives: proposing the first multi-stage bisection method in four (Appendix D) and arbitrary (Chapter 3) dimensions; enabling and guaranteeing for the first time newest vertex bisection advantages in three (Chapter 4) and arbitrary (Chapter 5) dimensions; and demonstrating the suitability of multi-stage bisection method for local refinement in arbitrary dimensions (Chapter 6). To demonstrate conformal bisection, we have used computer implementations, runtime checks, and formal proofs. As central findings, to enforce strongly compatible meshes suitable for local refinement, we have provided sufficient conditions to ensure mesh reflectivity and conformity.

To ensure reflectivity, we have proved that it is sufficient to sort the simplex vertices with a strict and total order of the mesh vertices, Chapter 5. We have used this sufficient condition to guarantee reflectivity in arbitrary dimensions for the newest vertex bisection and for the marked bisection. Remarkably, we might use this sufficient condition to guarantee reflectivity of other bisection methods. For instance, in the proposed 3D marked bisection with optimal similarity bound, the key idea would be to reinterpret the proposed strict and total order of the mesh edges. Actually, it is straightforward to check that the proposed edge ordering arises from the strict and total order of the mesh vertices. Thus, we can apply the reflectivity sufficient condition.

To ensure conformity, we have promoted checking conformity separately on faces

determined by mesh simplices and on faces generated by bisection, Chapter 6. We have used this check to demonstrate conformity for the marked bisection in arbitrary dimensions. As a perspective, we might use this approach to guarantee conformity for other bisection methods.

Moreover, we have constructively answered two open questions: Can a reflection structure be extracted from any mesh? Is it possible to perform multi-stage bisection in any mesh? First, we have extracted a reflection structure from any mesh in three or more dimensions. Thus, we have enabled the advantages of the newest vertex bisection for local refinement of complex geometry. Second, we have performed marked bisection in any mesh in four or more dimensions. Thus, we have provided methods that almost fulfill the advantages of the newest vertex bisection for local refinement of complex geometry.

The newest vertex bisection solution has been easier to guarantee and code than the marked bisection solution, and it features optimal stability and locality. On the contrary, the marked bisection solution has been easier to obtain than the newest vertex bisection solution. However, it is more difficult to code, and it only almost fulfills the optimal stability and locality properties. These differences are so because the newest vertex bisection can be understood as a multi-stage method with a first stage featuring zero refinements. Thus, the first answer also answers question two. However, the first stages of our marking bisection methods are not equivalent to the newest vertex bisection, and thus the second answer does not answer the first question.

The work carried out in this thesis leaves open some research activities that should be performed in the near future. First, although the proposed newest vertex bisection features theoretical and implementation advantages, it could be interesting to empirically compare it with the proposed marked bisection. Specifically, we would like to compare the resulting meshes for local refinement in arbitrary dimensions. Second, all the methods use the same implementation to detect hanging vertices. This operation might be optimized to accelerate bisection methods. We would like to explore approaches that only check those elements that have been refined, and different data structures. Third, propose and implement a parallelizable version of the two n-dimensional bisection methods. Fourth, couple the bisection method with a fast n-dimensional multi-grid method for the adaptive solution of high-dimensional partial differential equations. Fifth, explore initial re-orderings of the mesh vertices

and edges in such a manner that the number of bisected elements to maintain the conformal closure is reduced in the first bisection iterations. Sixth, add the capability of coarsening the proposed local refinement method.

Nevertheless, we have enabled mesh local refinement in arbitrary dimensions for unstructured conformal simplicial meshes. Specifically, we have proposed methods that guarantee (almost guarantee) the advantages for local refinement of newest vertex bisection on unstructured conformal meshes in more than two (three) dimensions.

In perspective, our n-dimensional bisection methods for unstructured conformal methods will enable adaptive applications on n-dimensional complex geometry, applications that model physical and economic phenomena. In this case, the complexity of the geometry would be handled by the flexibility of unstructured conformal meshes. Furthermore, on these general meshes, our refinement methods will exploit the advantages of the newest vertex bisection for n-simplicial adaption.

Appendix A

Lemmas relating sets and unions with simplices, concatenation and vertex sorting

---

In this appendix, we introduce three technical results that are used in the proof of Theorem 5.1.

Lemma A.1. *Let* $l = (l_0, \ldots, l_n)$ *and* $r = (r_0, \ldots, r_m)$ *be two simplices without vertices in common. Then,*

$$\texttt{set}(l \texttt{ + } r) = \texttt{set}(l) \cup \texttt{set}(\mathrm{r}).$$

Proof. We proof the result using the following chain of equalities

$\texttt{set}(l \texttt{ + } r) = \texttt{set}((l_0, \ldots, l_n) \texttt{ + } (r_0, \ldots, r_m))$ (By definition of l and r)
$= \texttt{set}((l_0, \ldots, l_n, r_0, \ldots, r_m))$ (By definition + in Equation (5.6))
$= \{l_0, \ldots, l_n, r_0, \ldots, r_m\}$ (By definition of set in Equation (5.1))
$= \{l_0, \ldots, l_n\} \cup \{r_0, \ldots, r_m\}$ (Because they do not share vertices)
$= \texttt{set}((l_0, \ldots, l_n)) \cup \texttt{set}((r_0, \ldots, r_m))$ (By definition of set in Equation (5.1))
$= \texttt{set}(l) \cup \texttt{set}(r)$. (By definition of l and r).

□

Lemma A.2. *Let σ be a simplex and consider its decomposition $\sigma = l \texttt{ + } (v) \texttt{ + } r$ as in Equation (5.7). Let κ be the opposite face to the vertex v. Then,*

$$\texttt{set}(\kappa) = \texttt{set}(l) \cup \texttt{set}(r).$$

Proof. Recall that after decomposing the sorted simplex as $\sigma = l \texttt{ + } (v) \texttt{ + } r$, we have that l, (v), and r do not have any vertex in common. Moreover, $v \notin \texttt{set}(\kappa)$. Therefore, the result arises from the following chain of equalities

$$\begin{aligned}
&\texttt{set}(\kappa) \\
&= \texttt{set}(\kappa) \cap \texttt{set}(\sigma) \text{ (Because } \kappa \text{ is a sub-simplex of } \sigma) \\
&= \texttt{set}(\kappa) \cap (\texttt{set}(l \texttt{ + } (v) \texttt{ + } r)) \text{ (Using } \sigma \text{ decomposition)} \\
&= \texttt{set}(\kappa) \cap (\texttt{set}(l) \cup \{v\} \cup \texttt{set}(r)) \text{ (By Lemma A.1)} \\
&= (\texttt{set}(\kappa) \cap \texttt{set}(l)) \text{ (Because the intersection distributes over union)} \\
&\quad \cup (\texttt{set}(\kappa) \cap \{v\}) \\
&\quad \cup (\texttt{set}(\kappa) \cap \texttt{set}(r)) \\
&= \texttt{set}(l) \cup \{\emptyset\} \cup \texttt{set}(r) \text{ (l and r and are sub-simplices of } \sigma, \text{ and } v \notin \kappa) \\
&= \texttt{set}(l) \cup \texttt{set}(r) \text{ ($\emptyset$ is neutral).}
\end{aligned}$$

□

Lemma A.3. *Let $l = (l_0, \ldots, l_n)$ and $r = (r_0, \ldots, r_m)$ be two sorted simplices without vertices in common such that $l_0 \ll \ldots \ll l_n \ll r_0 \ll \ldots \ll r_m$. Then,*

$$\texttt{simplex}(\texttt{set}(l) \cup \texttt{set}(r)) = l \texttt{ + } r.$$

Proof. The following chain of equalities proves the result

$$\begin{aligned}
&\texttt{simplex}(\texttt{set}(l) \cup \texttt{set}(r)) \\
&= \texttt{simplex}(\texttt{set}(l \texttt{ + } r)) \text{ (By Lemma A.1)} \\
&= \texttt{sort}(l \texttt{ + } r) \text{ (By definition of } \texttt{sort} \text{ in Equation (5.5))} \\
&= \texttt{sort}((l_0, \ldots, l_n) \texttt{ + } (r_0, \ldots, r_m)) \text{ (By definition of } l \text{ and } r) \\
&= \texttt{sort}((l_0, \ldots, l_n, r_0, \ldots, r_m)) \text{ (By definition of } \texttt{ + } \text{ in Equation (5.6))} \\
&= (l_0, \ldots, l_n, r_0, \ldots, r_m) \text{ (Because by hypothesis } l_0 \ll \ldots l_n \ll r_0 \ll \ldots \ll r_m) \\
&= (l_0, \ldots, l_n) \texttt{ + } (r_0, \ldots, r_m) \text{ (By definition of } \texttt{ + } \text{ in Equation (5.6))} \\
&= l \texttt{ + } r \text{ (By definition of } l \text{ and } r).
\end{aligned}$$

□

Appendix B

Visualization of pentatopic meshes

In order to devise, check, and illustrate the proposed refinement algorithm in the particular case of 4D meshes, we devise a simple tool to visualize 4D unstructured pentatopic meshes.

Our method is devised to exploit existent 3D visualization software which provides mature user interfaces to interact in real-time with 2D projections of the 3D meshes. To exploit these interfaces, we propose to slice unstructured 4D pentatopic meshes with an arbitrary 3D hyperplane and obtain a conformal 3D unstructured tetrahedral representation of the mesh slice that is ready to be read with standard 3D visualization tools.

Other methods to visualize 4D pentatopic meshes have been outlined in Caplan (2019); Neumüller and Steinbach (2011); Neumüller and Karabelas (2019). The main difference with our approach is that our resulting 3D visualization mesh is composed of tetrahedra, while the other methods provide meshes composed of polyhedra.

The input of the visualization algorithm is a 4D unstructured pentatopic mesh and a 3D hyperplane. First, we intersect the 4D mesh with the 3D hyperplane. To this end, we loop over the edges of the 4D mesh and intersect them with the hyperplane. If the intersection leads to a single point, we store the point. If the edge is contained inside the hyperplane, we store both endpoints of the edge. Otherwise, the edge does not intersect the hyperplane. The obtained points belong to the 3D hyperplane and

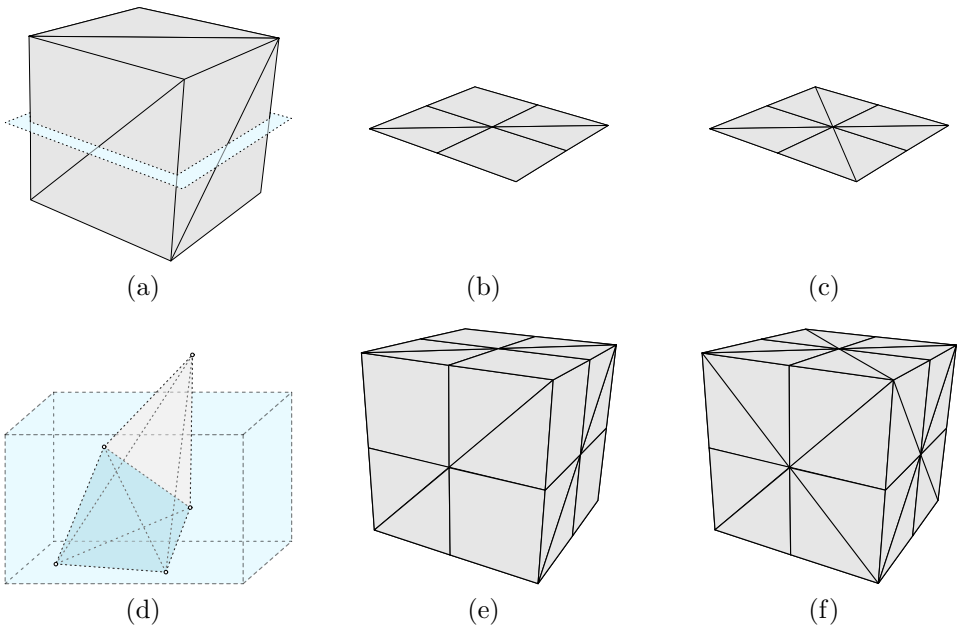


Figure B.1: The first row corresponds to the application of the visualization to the 3D tetrahedral mesh that generate a 2D triangular visualization mesh. We slice a tetrahedral mesh (a) with a 2D plane, obtaining a polygonal mesh (b). Then, we generate a 2D triangular mesh (c). Analogously, the second row corresponds to the visualization of a 4D pentatopic mesh. First, we slice a 4D pentatopic mesh with a 3D hyperplane (d), obtaining a polyhedral mesh B.1(e). Then, we decompose the obtained polyhedral mesh into tetrahedral mesh (f).

therefore, we map them using an orthonormal base of the 3D hyperplane using three coordinates.

If the 4D mesh contains a field, we perform a linear interpolation to obtain the value of the field at the intersection points. Thus, the visualization mesh also contains the slice of the field.

Once we have the intersection points, we loop over the pentatopes of the 4D mesh. For each pentatope, we recover the intersection points of its local edges. If the points define a valid volume in the 3D space, we perform a local 3D Delaunay using the intersection points of the pentatope. The main advantage of applying the Delaunay method is that we do not need to generate all the possible templates for all the possible configuration of points. We only need to ensure that the points lead to a valid volume in the 3D space. Finally, we gather all the local tetrahedralizations into a single 3D mesh. In Figure B.1 we illustrate the proposed visualization method applied to a 3D and 4D case.

Appendix C

Conformity, reflectivity, and mesh renumbering

C.1 Conformity check

The main goal of this appendix is to propose an algorithm to check the conformity of the meshes generated by the local refinement algorithm. According to Theorem 2.1, to check that a mesh $\mathcal{T}$ is conformal, we only check conformity through mesh faces. The main idea of the algorithm is to use the multi-indices of the vertices of the original and bisected meshes.

Let $\mathcal{T}_0$ be a conformal simplicial mesh and $\mathcal{M}_0$ the set of simplices to bisect. Consider the mesh $\mathcal{T}_1$ obtained after refining the set of simplices $\mathcal{M}_0$ of $\mathcal{T}_0$ using the local refinement algorithm. Since we bisect edges during the refinement process, we have two types of multi-indices in $\mathcal{T}_1$: the original vertices $[v_i]$ of $\mathcal{T}_0$, and the new vertices $[v_{i_1}, v_{i_2}, \ldots, v_{i_k}]$, which are generated during the bisection process. We know that $\mathcal{T}_0$ is conformal, and we want to check that $\mathcal{T}_1$ is conformal, too.

The idea is devised in two steps. The first one is to check that there are no faces shared by more than two simplices. The second one is to check that all the boundary faces of $\mathcal{T}_1$ are contained in the boundary faces of $\mathcal{T}_0$. The boundary check allows us to ensure that we do not create holes during the bisection process. For that purpose, we use a dictionary data structure to define a map between a sorted face κ and the simplices that contain it. We propose to create two dictionaries: a dictionary of inner

Algorithm C.1 Generation of a dictionary of faces

input: `Mesh` $\mathcal{T}$
output: `Dictionary` $\mathcal{I}$, `Dictionary` $\mathcal{B}$

1: **function** `getFaces`($\mathcal{T}$)
2: $\mathcal{D} = \{\}$; $\mathcal{I} = \{\}$; $\mathcal{B} = \{\}$ ▷ Dictionaries of all the faces, inner faces and boundary faces, respectively.
3: **for** σ **in** $\mathcal{T}$ **do**
4: $(\mathbf{v}_{i_0}, \mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_n}) = \texttt{sort}(\sigma)$
5: **for** $j = 0, 1, \ldots, n$ **do**
6: $\kappa_j = \texttt{oppositeFace}((\mathbf{v}_{i_0}, \mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_n}), \mathbf{v}_{i_j})$
7: **if** $\mathcal{D}[\kappa_j] = \emptyset$ **then**
8: $\mathcal{D}[\kappa_j] = \sigma$
9: **else**
10: $\mathcal{D}[\kappa_j] = \texttt{append}(\mathcal{D}[\kappa_j], \sigma)$
11: **end if**
12: **end for**
13: **end for**
14: **for** κ **in** `keys`($\mathcal{D}$) **do**
15: **if** `length`($\mathcal{D}[\kappa]$) $= 1$ **then**
16: $\mathcal{B}[\kappa] = \mathcal{D}[\kappa]$
17: **else if** `length`($\mathcal{D}[\kappa]$) $= 2$ **then**
18: $\mathcal{I}[\kappa] = \mathcal{D}[\kappa]$
19: **else**
20: **Error**: A face cannot be shared by more than two simplices.
21: **end if**
22: **end for**
23: **return** $\mathcal{I}$, $\mathcal{B}$
24: **end function**

faces, $\mathcal{I}$; and a dictionary of boundary faces $\mathcal{B}$. Thus, for a given sorted face κ, the operation $\mathcal{I}[\kappa]$ and $\mathcal{B}[\kappa]$ return the simplices, or simplex, that contain the sorted face κ.

The proposed method to generate $\mathcal{I}$ and $\mathcal{B}$ is shown in Algorithm C.1. First, we do a loop over the simplices of the mesh $\mathcal{T}$, see Line 3, and sort them using the lexicographic order, see Line 4. Then, we compute the faces of the sorted simplex using the opposite faces method, a procedure that for a given simplex σ and a given vertex $[v]$, returns the opposite face κ to the vertex $[v]$ inside the simplex σ, see Line 6. Since the vertices of the simplex are ordered, the vertices of each face are also ordered. After that, if the sorted face κ is not inside the dictionary $\mathcal{D}$, we add it

Algorithm C.2 Checking of conformity

input: `Mesh` $\mathcal{T}_1$, `Conformal Mesh` $\mathcal{T}_0$
output: `Bool isConformal`

1: **function** `isMeshConformal`($\mathcal{T}_1$, $\mathcal{T}_0$)
2: $\quad\mathcal{I}_0, \mathcal{B}_0 =$ `getFaces`($\mathcal{T}_0$)
3: $\quad\mathcal{I}_1, \mathcal{B}_1 =$ `getFaces`($\mathcal{T}_1$)
4: $\quad$**for** κ_1 **in** `keys`($\mathcal{B}_1$) **do**
5: $\quad\quad([v_{i_0}], [v_{i_1}], \ldots, [v_{i_{n-1}}]) =$ `getFathersVertices`(κ_1)
6: $\quad\quad$**if** $([v_{i_0}], [v_{i_1}], \ldots, [v_{i_{n-1}}])$ **not in** `keys`($\mathcal{B}_0$) **then**
7: $\quad\quad\quad$**return** `false`
8: $\quad\quad$**end if**
9: $\quad$**end for**
10: $\quad$**return** `true`
11: **end function**

and assign as a value the simplex σ, see Line 8. If the sorted face κ is already in the dictionary $\mathcal{D}$, we append the simplex σ to the value $\mathcal{D}[\kappa]$, see Line 10. Now we have created a dictionary $\mathcal{D}$ that contains all the faces of the mesh $\mathcal{T}$. Thus, we can determine if a face κ is a boundary face or an inner face checking how many values has the sorted face κ inside $\mathcal{D}$, see Lines from 14 to 22. If some face appears more than two times, the algorithm stops because a face cannot be shared by more than two simplices. Finally, we return as output the dictionaries $\mathcal{I}$ and $\mathcal{B}$. We recall that we sort lexicographically all the faces of $\mathcal{T}$ to create the keys of the dictionaries $\mathcal{I}$ and $\mathcal{B}$. The order of those faces inside the simplices that contain them could be different from the lexicographical order used to define the keys of the dictionaries.

After the definition of the dictionaries $\mathcal{I}$ and $\mathcal{B}$, we propose an algorithm to check the conformity of the mesh, checking that the boundary faces of $\mathcal{T}_1$ are contained inside the boundary faces of $\mathcal{T}_0$. The procedure is depicted in Algorithm C.2. Thus, for a given conformal mesh $\mathcal{T}_0$ and a mesh $\mathcal{T}_1$, the function that check if a mesh is conformal, returns a boolean value depending if the boundary of $\mathcal{T}_1$ is contained, or not, inside the boundary of $\mathcal{T}_0$. That is, if $\mathcal{T}_1$ is conformal to $\mathcal{T}_0$. First, we compute the boundary dictionaries of $\mathcal{T}_0$ and $\mathcal{T}_1$ respectively, see Lines 2 and 3. When we compute the dictionaries, we are checking that there are no faces that appear more than two times. After that, for each sorted boundary face κ_1 of $\mathcal{T}_1$, we obtain its parent face κ_0 of $\mathcal{T}_0$. To do it so, we first collect in a list all the indices in all the multi-indices of κ_1. Then, we extract the unique indices of this list. We should obtain a list of length n, $([v_{i_0}], [v_{i_1}], \ldots, [v_{i_{n-1}}])$. This list of vertices is converted to a face κ_0.

Algorithm C.3 Checking of reflected neighbors

input: `ConformalMesh` $\mathcal{T}$
output: `Bool`

1: **function** `isReflected`($\mathcal{T}$)
2: $\mathcal{I}, \mathcal{B} =$ `getFaces`($\mathcal{T}$)
3: **for** $\kappa \in$ `keys`($\mathcal{I}$) **do**
4: $([v_0], [v_1], \ldots, [v_{n-1}]) = \kappa$
5: $\sigma_1, \sigma_2 = \mathcal{I}[\kappa]$
6: $([v_{1,0}], [v_{1,1}], \ldots, [v_{1,n}]) = \sigma_1$
7: $([v_{2,0}], [v_{2,1}], \ldots, [v_{2,n}]) = \sigma_2$
8: $[w_1] =$ `oppositeVertex`(σ_1, κ)
9: $[w_2] =$ `oppositeVertex`(σ_2, κ)
10: $\kappa_1 =$ `remove`(σ_1, $[w_1]$)
11: $\kappa_2 =$ `remove`(σ_2, $[w_2]$)
12: $([v_{i_0}], [v_{i_1}], \ldots, [v_{i_{n-1}}]) = \kappa_1$
13: $([v_{j_0}], [v_{j_1}], \ldots, [v_{j_{n-1}}]) = \kappa_2$
14: **for** $k = 0, 1, \ldots, n-1$ **do**
15: **if** $v_{i_k} \neq v_{j_k}$ **then**
16: **return** `false`
17: **end if**
18: **end for**
19: **end for**
20: **return** `true`
21: **end function**

Since all the vertices are of length one, we check if the face belongs to the boundary of $\mathcal{T}_0$ by checking the dictionary of boundary faces $\mathcal{B}_0$. Otherwise, the boundary face κ_1 is not contained inside a boundary face of $\mathcal{T}_0$, and the mesh is not conformal.

C.2 Reflectivity check

To check that $\mathcal{Q}_n$ is reflected, we have to check that all neighboring simplices are reflected neighbors. To do that, we use Algorithm C.3. First, we obtain the dictionary $\mathcal{D}$ of the interior faces of $\mathcal{Q}_n$, see Line 2. For each inner face $\kappa \in \mathcal{Q}_n$, the operation $\mathcal{I}[\kappa]$ return the two neighboring simplices σ_1 and σ_2 that share κ. The face $\kappa = ([v_0], \ldots, [v_{n-1}])$, with $v_0 < v_1 < \ldots < v_n$, has lexicographic order in $\mathcal{D}$, but it may have a non-shared order inside σ_1 and σ_2. After obtaining the σ_1 and σ_2, we obtain the different vertices $[w_1]$ and $[w_2]$ between the simplices σ_1 and σ_2 and the face κ, respectively, see Lines 8 and 9. Then, we obtain the faces κ_1 and κ_2 inside σ_1 and

σ_2 with the order in which they appear inside each simplex, see Lines 10 and 11. We recall that κ_1, κ_2, and κ are composed of the same vertices, nevertheless, the vertices may appear in a different order. What we want to check is that κ_1 and κ_2 have the same order of vertices. To this end, we loop on the faces κ_1 and κ_2, see Lines 14–18, checking that they have the same vertices and in the same order.

C.3 Mesh renumbering

After obtaining a locally refined mesh, we renumber the resulting multi-ids to have new multi-ids of length one and thus, reduce the memory usage. In Line 4 of local refinement algorithm, see Algorithm 2.3, we perform a renumber of the vertices of the generated mesh. Our algorithm identifies the vertices of the mesh with multi-ids, see Section 2.2. At the end of the local refinement algorithm, we perform a renumber of the multi-ids of $\mathcal{T}_{k+1}$ to multi-ids of length one. To that purpose, we sort lexicographically all the multi-ids of $\mathcal{T}_{k+1}$ obtaining a sorted list of multi-ids. Then, we assign to the i-th multi-id of the sorted list the new multi-id of length one $[i]$. We have to perform that renumber of multi-ids in all the data structures where the multi-ids appear.

Appendix D

Local bisection for conformal refinement of unstructured 4D simplicial meshes

---

We append the paper Belda-Ferrín et al. (2019), presented at 27th International Meshing Roundtable at Albuquerque, United States.

Local bisection for conformal refinement of unstructured 4D simplicial meshes

Guillem Belda-Ferrín, Abel Gargallo-Peiró* and Xevi Roca

Abstract We present a conformal bisection procedure for local refinement of 4D unstructured simplicial meshes with bounded minimum shape quality. Specifically, we propose a recursive refine to conformity procedure in two stages, based on marking bisection edges on different priority levels and defining specific refinement templates. Two successive applications of the first stage ensure that any 4D unstructured mesh can be conformingly refined. In the second stage, the successive refinements lead to a cycle in the number of generated similarity classes and thus, we can ensure a bound over the minimum shape quality. In the examples, we check that after successive refinement the mesh quality does not degenerate. Moreover, we refine a 4D unstructured mesh and a space-time mesh (3D + 1D) representation of a moving object.

1 Introduction

In the last three decades refinement of 2D and 3D unstructured simplicial meshes [1–14], based on red/green refinement [1–7] and bisection [8–14], has been shown to be a key ingredient on efficient adaptive loops. Although one could expect the same in 4D, a case of special interest for space-time adaption, this line of research has not been extensively explored.

For our space-time applications, we are interested in conformal bisection methods since they are really well suited to implement fast geometrical multi-grid conformal solvers. Moreover, bisection methods have ensured either a maximum number of generated similarity classes [11–13] or a minimum lower quality bound over the generated elements after successive refinements [8–10, 14]. Regarding 4D refinement, only a non-conformal local refinement method for pentatopic meshes has

Computer Applications in Science and Engineering, Barcelona Supercomputing Center, 08034 Barcelona, Spain.
*Corresponding author e-mail: abel.gargallo@bsc.es

been proposed [15]. Unfortunately, existent conformal 4D (nD) bisection methods with a bound over the number of generated similarity classes [11, 12] cannot be applied to general unstructured meshes.

The main contribution of this work is to propose a local bisection procedure, with a bound over the number of generated similarity classes, for conformal refinement of 4D unstructured simplicial meshes. Specifically, we propose a recursive refine to conformity procedure, in two stages, based on marking bisection edges on different priority levels (Sec. 3.1). The marking procedure allows classifying the pentatopes in different types (Sec. 3.2) and hence, determining different refinement templates (Sec. 4), in an analogous manner to the 3D bisection method proposed in [13].

The refinement method is composed of two stages (Sec. 4). Two successive applications of the initial stage of the bisection strategy (Sec. 4.1), based on the proposed element classification, ensure that any initial 4D unstructured simplicial mesh can be conformingly refined. After the two initial refinements our recursive refine to conformity strategy switches to the second stage (Sec. 4.2). This final stage is analogous to Maubach's algorithm, when it is successively applied to a single pentatope. Therefore, we can ensure a bound over the number of generated similarity classes. Thus, the minimum quality of the refined mesh is bounded, independently of the number of performed refinements. The main advantage and difference of our method when compared to Maubach's algorithm [11] is the first stage of the method, which allows the application of the method to any 4D unstructured simplicial mesh.

In all the examples (Sec. 5), we show that the proposed methodology leads to a periodic evolution of the minimum element quality (shape quality measure [16]) illustrating the lower bound of the quality through successive refinement. We first illustrate how to check that an implementation of the proposed method is valid by successively refining a pentatope. With our implementation, we show that the proposed bisection technique can be used to refine general unstructured 4D meshes. Finally, we also illustrate our application of interest, the refinement of a 4D mesh corresponding to a space-time representation, with varying resolution, of the temporal evolution of a 3D moving object.

2 Preliminaries

In this section, we state some preliminary notions required for the rest of this work. First, we detail how a pentatope in four dimensions is represented in the 2D plots of this paper. Second, we state the definition of bisection and finally, we introduce the strategy used in this work to refine a given mesh through edge bisection.

The element type considered in this work is the *pentatope* (4D simplex) which is defined as the convex hull of a set of 5 points $\{x_0, x_1, x_2, x_3, x_4\}$ in $\mathbb{R}^4$. To represent a given pentatope (4D) in the plane (2D), we focus on a perspective where the edges connecting the vertices have a minimal number of edge crossings. Herein, the pentatope $[x_0x_1x_2x_3x_4]$ is displayed plotting the edges of the tetrahedron $[x_0x_1x_2x_3]$ and the edges that connect the vertices of the tetrahedron $[x_0x_1x_2x_3]$ with the extra ver-

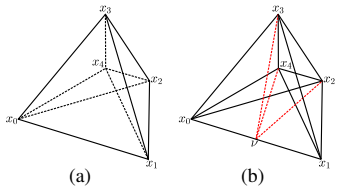


Fig. 1 (a) Three dimensional representation of a pentatope $[x_0x_1x_2x_3x_4]$, where the fifth vertex x_4 is plotted in $\mathbb{R}^3$ inside the tetrahedron $[x_0x_1x_2x_3]$. (b) Potential edges $[vx_2]$, $[vx_3]$ and $[vx_4]$ of the bisection of the pentatope $[x_0x_1x_2x_3x_4]$.

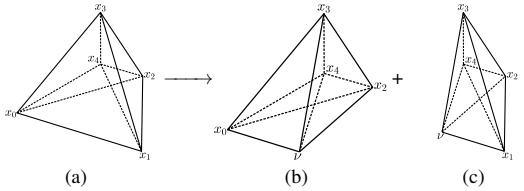


Fig. 2 Bisection of a pentatope (a) into two children (b) and (c).

tex x_4 located in the center of the tetrahedron, see Figure 1(a). This representation is used to display the edge marking procedure for bisection proposed in this work. The boundary of a pentatope is formed by 5 tetrahedra: the outer tetrahedron $[x_0x_1x_2x_3]$ and the four inner tetrahedra $[x_0x_1x_2x_4]$, $[x_0x_1x_4x_3]$, $[x_0x_2x_3x_4]$ and $[x_1x_2x_3x_4]$.

Once detailed the representation of a given pentatope, we particularize the definition of bisection to 4D simplicial elements. In particular, for a given pentatope σ with vertices $[x_0x_1x_2x_3x_4]$, the element vertices are reordered so that the refinement edge is $[x_0x_1]$. Let v be the midpoint of $[x_0x_1]$. The bisection of σ by $[x_0x_1]$ corresponds to removing the element $[x_0x_1x_2x_3x_4]$ and generating two new elements by joining v with the tetrahedral faces $[x_0x_2x_3x_4]$ and $[x_1x_2x_3x_4]$.

We highlight that the tetrahedral face $[vx_2x_4x_3]$ is shared between the two children. This shared face has three inherited edges ($[x_2x_3]$, $[x_2x_4]$ and $[x_3x_4]$) and three new edges ($[vx_2]$, $[vx_3]$ and $[vx_4]$). We denote the new edges of the shared face as *potential edges* of the initial element. These potential edges are displayed in Figure 1(b) colored in red. This definition is required in Section 4.2 to characterize the proposed mesh refinement templates.

Finally, we introduce the algorithm proposed in this work to refine a given mesh by edge bisection. This algorithm uses a refine to conformity strategy similar to the 3D refinement method proposed in [13]. Given a marked mesh M and a set of elements to refine S, the mesh is refined according to Algorithm 1. In this algorithm, while there is not an empty set of elements to refine, Line 2, the mesh is refined as follows. In Line 3, the process `BisectPentatopes` bisects each pentatope in S:

Procedure 1 Refinement of a mesh ensuring conformity.

Input: Marked mesh M
Output: Marked mesh M'
1: **function** REFINETOCONFORMITY(M,S)
2: **if** $S \neq \emptyset$ **then**
3: $\bar{M} = \texttt{BisectPentatopes}(M,S)$
4: $S = \{\sigma \in M \,|\, \sigma \text{ has a hanging node}\}$
5: $M' = \texttt{RefineToConformity}(\bar{M})$
6: **else**
7: $M' = M$
8: **end if**
9: **end function**

$$\texttt{BisectPentatopes}(M,S) = (M \setminus S) \cup \bigcup_{\sigma \in S} \texttt{Bisect}(\sigma), \tag{1}$$

where `Bisect` performs the element bisection taking into account the element marks (refinement edge) and sets the proper marks to the two generated elements. In Sections 3 and 4 we will present the marking procedures proposed in this work for pentatopic meshes, and the marks that are assigned to the two children.

Following, in Line 4 the set of elements to refine in the next step is set as the elements with hanging nodes. In Line 5 the Algorithm `RefineToConformity` is called recursively. These recursive calls are continued until there are no more elements with hanging nodes in the mesh. We show that the marking processes presented in Sections 3 and 4 lead to a conformal mesh.

3 Edge marking and element classification for compatible refinement

In this section, we first present in Sec. 3.1 an edge marking process compatible between neighboring elements for conformal mesh refinement. Next, in Sec. 3.2 we present a classification of the elements of the mesh depending on the marks assigned to their edges.

3.1 Edge marking for compatible refinement

In this work, we use a marking procedure organized by levels to determine the priority of the bisection edges used during the element refinement. Following, we present a procedure to mark the edges of the pentatopes of a conformal mesh. These marks are devised to ensure that for a given face shared between two pentatopes, successive bisection of surrounding elements determines the same mesh from both sides of the shared face. Hence, this ensures mesh conformity along the bisection process.

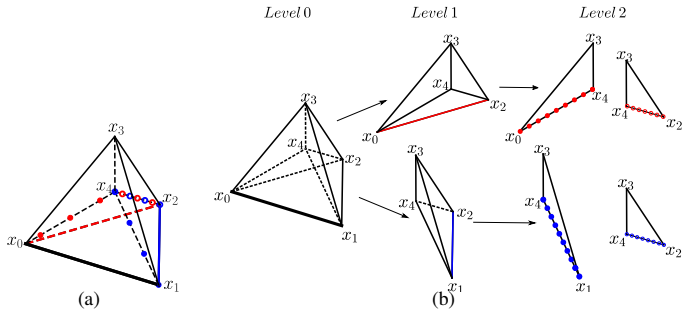


Fig. 3 (a) Marked pentatope and (b) marking diagram process at different levels.

We define three levels of marks in a pentatope. The level 0 features one edge, which corresponds to the refinement edge of the current pentatope. The level 1 features two edges, which correspond to the refinement edges of the two children of the first pentatope. Finally, the level 2 features four edges, which correspond to the refinement edge of the four grandchildren of the original pentatope.

Herein, to determine the marks assigned to each of the edges of the element, we prioritize the edges in terms of their length with a well-defined tie-breaking rule. For a given element we define its *consistent bisection edge* as the edge of longest length and lowest global index. The lowest global index is a tie-breaking rule that ensures that if there exist multiples edges with the same length, we select as the longest edge always the same one, independently of the order in which the edges are compared. In particular, with this tie-breaking rule we ensure that the edges of a common face between two adjacent pentatopes are marked in the same manner from the two of them. We remark that the longest edge is considered in the *consistent bisection edge* sorting rule since it is an heuristic to enforce better element quality. Nevertheless, it is not a key ingredient to ensure conformal mesh refinement. For instance, just by sorting the mesh edges using their global index would also lead to a valid consistent sorting rule.

Following, we detail the marking process for a given pentatope $[x_0x_1x_2x_3x_4]$. The process consists of three steps, illustrated in Figure 3(b):

1. Marked edge of level 0: *consistent bisection edge* of the pentatope $[x_0x_1x_2x_3x_4]$. In the bisection process, the marked edge of level 0 corresponds to the bisection edge of the element. In this work, the marked edges of level 0 are plotted with a thick black line, see the first column of Figure 3(b).
2. Marked edges of level 1: the two marked edges of level 1 are determined as the *consistent bisection edge* of the tetrahedra defined by $[x_1x_2x_3x_4]$ and $[x_0x_2x_3x_4]$. These two tetrahedra are indeed the opposite tetrahedral faces of the pentatope with respect to x_0 and x_1, respectively. These two tetrahedral faces are the faces of the original pentatope preserved in each child. The two marked edges of level 1 correspond to the bisection edges of the two children of the current element.

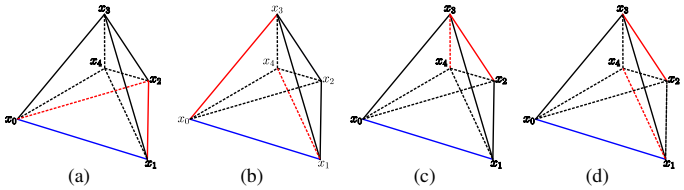


Fig. 4 Type of edge relations between the edges on level $l+1$ (red) and the edge of level l (blue): (a) P, (b) A, (c) O, and (d) M.

A particular configuration of the marked edges of level 1 is illustrated in the second column of Fig. 3(b). The marked edges of level 1 associated to the first and second node of the bisection edge are colored in red and blue, respectively.

3. Marking edges of level 2: the four marked edges of level 2 are determined as the *consistent bisection edge* of the opposite faces of the marked edges of level 1 in the tetrahedra $[x_1x_2x_3x_4]$ and $[x_0x_2x_3x_4]$. The four marked edges of level 2 correspond to the bisection edges of the four grandchildren of the current marked element. In the third column of Figure 3(b), we illustrate a particular configuration of the marked edges of level 2, coloring them with the same color of the associated marked edge of level 1. In addition, the edge associated to the first node of the marked edge of level 1 is plotted with fully colored circles, and the other edge is plotted with empty circles.

Figure 3(a) illustrates the resulting marked element for the test example of the marking procedure of Fig. 3(b). We highlight an edge, for instance $[x_2x_4]$ in Fig. 3(a), can have two marks once all the marks are displayed on the initial pentatope. These two marks indicate that this edge has been marked from both of the faces that remain after bisection. To differentiate them, we have used blue and red colors. After bisecting a marked pentatope, the marked edges of the two children have to be determined.

Remark 1 (Inheritance of marks). The marked edges of level 1 and 2 of the parent shift marks in the corresponding children and become the marked edges of level 0 and 1 of the two children, respectively. However, it is not straight-forward to determine the marked edges of level 2 from the parent marks. In Section 4 two methods are proposed to determine them.

3.2 Classification of marked pentatopes

In this section, we present a classification into different types of a pentatope resulting from the marking process detailed in Section 3.1. Several types of pentatopes are obtained depending on the marks assigned to their edges. Before detailing the

classification, we introduce four definitions that state how the marked edges of level $l+1$ are located with respect to the associated marked edge of level l for $l = 0, 1$.

We propose a classification for different pentatope types, according to the configuration of the marked edges at the different levels. This classification is an extension of the different tetrahedron types proposed in [13], where only two levels of marked edges are required. Figure 4 illustrates the four different configurations between two levels of marked edges, coloring the two marked edges of level $l+1$ with red color and the marked edge of level l with dark blue color:

- Type P (Planar): the two marked edges of level $l+1$ are coplanar with the marked edge of level l, *i.e.*, the three edges are connected defining a triangle. In Figure 4(a) an example of edges of type P is illustrated.
- Type A (Adjacent): each marked edge of level $l+1$ has a common vertex with the marked edge of level l but the two edges of level $l+1$ do not have any common vertex. In Figure 4(b) an example of edges of type A is illustrated.
- Type O (Opposite): the marked edges of level $l+1$ of the opposite faces of the marked edge of level $l+1$ do not intersect the marked edge of level l. In Figure 4(c) an example of edges of type O is illustrated. We highlight that a possible configuration of edges of type O is that the two edges of level $l+1$ are overlapped. For instance, the edge $[x_2x_3]$ could be the marked edge of level $l+1$ for the two faces opposite to the edge of level l.
- Type M (Mixed): the marked edges of level $l+1$ of just one of the opposite faces have a common vertex with one marked edge of level l. In Figure 4(d) an example of edges of type M is illustrated. We highlight that it is possible that the marked edges of level $l+1$ have a common vertex between them. For example, the marked edges of level $l+1$ could be $[x_1x_4]$ and $[x_4x_3]$.

Herein, in a pentatope we have marked edges of level 0, 1 and 2. We denote by α the edge type determined by how marked edges of level 1 are located with respect to the marked edge of level 0. Additionally, we denote by β and γ the edge relation type between the marked edges of level 2 and the marked edge of level 1. In this manner, a marked pentatope is classified into a type of the form $\alpha_{\beta\gamma}$.

In Figure 5, we illustrate three different types of marked pentatopes. First, Figure 5(a) illustrates a pentatope of type P_{PP}. In particular, the marked edge of level 0 (bisection edge) configures a triangular face together with the marked edges of level 1. Thus, the first index, α, of the element is P. Next, each marked edge of level 1 defines also a triangular face with the corresponding marked edges of level 2, determining β and γ equal to P. Hence, the element is of type P_{PP}.

Analogously, for the element illustrated in Figure 5(b) we detail the same process. For this element, α is A since the red and blue edges share a node with the bisection edge, but they do not share any node between them. In addition, β and γ are equal to P, since each of the blue and red edges determines a triangular face with the corresponding blue and red circled edges. Thus, this element is of type A_{PP}. Similarly, we can conclude that the element illustrated in Figure 5(c) is of type A_{AA}.

After bisecting the element, the bisection edge $[x_0x_1]$ is split into two edges, $[x_0v]$ and $[vx_1]$, and thus this edge is not present in any of the children. However, the two

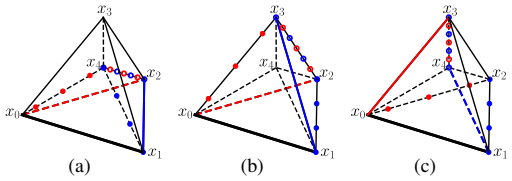


Fig. 5 Three different types of marked pentatopes: (a) P_{PP}, (b) A_{PP}, and (c) A_{AA}.

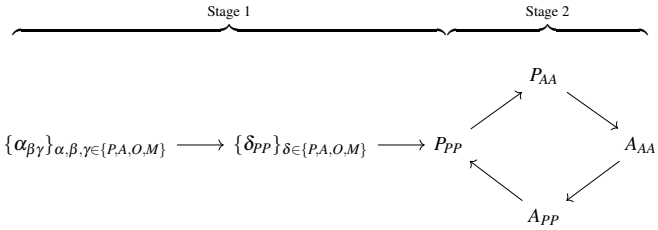


Fig. 6 Refinement process for a pentatope of type $\alpha_{\beta\gamma}$, where $\alpha,\beta,\gamma \in \{P,A,O,M\}$.

adjacent tetrahedral faces to this edge are preserved. Specifically, the face $[x_0x_2x_3x_4]$ is inherited by the child that preserves node x_0 of the bisection edge, and the face $[x_1x_2x_3x_4]$ is inherited by the child that preserves node x_1. Following we detail which marks of the parent pentatope are preserved after its bisection and how these marks are inherited by the two children.

Remark 2 (Inheritance of element type). Hence, after bisecting a marked pentatope of type $\alpha_{\beta\gamma}$, where α,β,γ can be $\{P,A,O,M\}$, the obtained children inherit the marked edges of level 1 and 2 of the parent. These marks become the marked edges of level 0 and 1 of the children, see Remark 1. Thus, one child inherits the edge relation type β and the other child the relation type γ. However, the marked edges of level 2 are not determined. Depending on the edges that are selected to be the marked of level 2, the type of element of the children will be $\beta_{\beta_1\beta_2}$ and $\gamma_{\gamma_1\gamma_2}$, where $\beta_1,\beta_2,\gamma_1,\gamma_2 \in \{P,A,O,M\}$.

4 A refinement algorithm for 4D unstructured simplicial meshes with bounded number of similarity classes

In this section, we detail a new procedure composed by two stages for refinement of any 4D unstructured simplicial mesh. Given a mesh, we first mark it using the procedure stated in Section 3.1, and following, we classify the elements into the different types stated in Section 3.2. Next, given a marked element to be refined, the bisec-

Procedure 2 Element refinement with global mesh conformity

Input: Pentatope σ, set of marked edges m_σ, descendant level k.
Output: Pentatopes σ_1 and σ_2, set of marked edges m_{σ_1} and m_{σ_2}, descendant level k'

1: **function** BISECT(σ , m_σ, k)
2: $\sigma_1, \sigma_2 =$ `BisectPentatope`(σ, m_σ)
3: $m_{\sigma_1}, m_{\sigma_2} =$ `inheritMarksFromFather`(m_σ)
4: **if** $k < 2$ **then**
5: $m_{\sigma_1}, m_{\sigma_2} \leftarrow$set marked edges of level 2 using Stage 1 from Sec. 4.1
6: **else**
7: $m_{\sigma_1}, m_{\sigma_2} \leftarrow$set marked edges of level 2 using Stage 2 from Sec. 4.2
8: **end if**
9: $k' = k+1$
10: **end function**

tion of this element is performed according to Algorithm 2 and the diagram in Fig. 6. Algorithm 2 is used as bisection procedure in the `BisectPentatopes` function, Eq. (1), from the mesh refinement strategy `RefineToConformity` presented in Algorithm 1.

The two stages bisect a given marked element according to the bisection edge, Line 2 from Alg. 2, and following, according to Remarks 1 and 2, the marked edges of level 0 and 1 of the children are determined from the marked edges of level 1 and 2 of the parent, Line 3. The difference between the two stages is the process to set the marked edges of the children. The marks determine the type of the generated element and at the same time, how the children will be bisected through successive refinement.

The first two times that the element is bisected (Stage 1), Line 4 of Alg. 2, the marks of level 2 of the children are determined using Sec. 4.1. A child generated with one application of Stage 1 is of type δ_{PP}, being δ any edge relation type. No element enters to Stage 2 before two refinements, and once it is bisected twice in Stage 1, in Section 4.1 we show that it is of type P_{PP}. Then, from the second refinement and on, Line 7, Stage 2 is activated, see Section 4.2. In Section 4.3 the properties of the two-stage method are presented.

4.1 Stage 1: refinement from any unstructured marked mesh to P_{PP} elements

Stage 1 determines the marked edges of level 2 following the ideas of the marking strategy presented in Section 3.1. From the marking diagram presented in Figure 3(b) we observe that two of the edges of the triangular face of the third column remain unmarked in the parent. After the element is bisected, these edges are still present in the tetrahedral faces of the children. These edges can be enforced to be the marked edges of level 2 of the children. This decision is consistent by construction between adjacent elements since it is performed on the face shared between these

elements. This approach to determine the marked edges of level 2 of the children leads to a conformal refinement procedure.

Remark 3 (Refinement towards P_{PP} elements). Given an element of type $\alpha_{\beta\gamma}$ for $\alpha,\beta,\gamma \in \{P,A,O,M\}$, the application of two refinements of Stage 1 leads to elements of P_{PP}, see Figure 6.
To show this, we first focus on the initial refinement step. From Remark 2 the children will be $\beta_{\beta_1\beta_2}$ and $\gamma_{\gamma_1\gamma_2}$, where $\beta_1,\beta_2,\gamma_1,\gamma_2 \in \{P,A,O,M\}$ depend on how the marked edges of level 2 are located with respect to the marked edges of level 1. By construction (see Fig. 3(b)), the marked edges of level 2 have been chosen on the same triangular face of the corresponding marked edge of level 1. Thus, the new marked edges of level 2 are coplanar with the marked edges of level 1 for each child and their edge relation is of type P. Hence, by setting these edges as marked edges of level 2 of the children, we obtain two children of type β_{PP} and γ_{PP}, respectively. Applying this marking strategy again, the grandchildren of the original pentatope are of type P_{PP}.

Although the marking process is consistent between adjacent elements by construction and the marks of level 2 are chosen consistently with the marking process, following we analyze all the possible neighboring configurations between two marked elements to illustrate that the stated bisection procedure is conformal.

Remark 4 (Conformal refinement). Given two neighbor marked elements, when the shared face is bisected from the two sides, it is bisected by the same edge. That is, the interface between the children of the two elements is still conformal. We analyze three different configurations of the two elements:

- First, let us assume that both elements share a face that contains their *consistent bisection edge*. This edge must be the same for each one of the elements, since in particular, it is the *consistent bisection edge* of the face. Then, it is clear that they are refined by that edge and that the new interface is conformal.
- Second, let us assume that the shared face does not contain the *consistent bisection edge* in any of the two adjacent elements. Following the stated marking procedure, the shared tetrahedral face is marked in the second column of Figure 3(b), containing the marked edges of level 1. Thus, in the first refinement of the elements, the face is not refined and the interface is still conformal. Next, when we perform the second refinement, the shared face is refined by the same edge from the two elements, ensuring a conformal bisection.
- Finally, the third case to be analyzed is when the face contains the *consistent longest edge* of the pentatope in one element, but does not contain the *consistent longest edge* of the adjacent pentatope. After refining once the elements, the mesh is not conformal, since the face is bisected from one of the elements, but is not bisected from the other one. However, the element that has not bisected the initially shared face, does bisect it after the second refinement, since the *consistent longest edge* of the adjacent pentatope is specifically the *consistent longest edge* of the shared face, and thus it is marked in the level 1 of the second element. Hence, after two iterations the mesh is already conformal.

Procedure 3 Bisection of a simplex from Maubach [11].

Input: Tagged n-simplex σ .
Output: Tagged n-simplices σ_1 and σ_2.
1: **function** BISECTSIMPLEXMAUBACH(σ)
2: Set $d' = \begin{cases} d-1, & d>1 \\ n, & d=1 \end{cases}$
3: Create the new vertex $z = \frac{1}{2}(x_0 + x_d)$.
4: Set $\sigma_1 = ((x_0, x_1, \ldots, x_{d-1}, z, x_{d+1}, \ldots, x_n), d')$.
5: Set $\sigma_2 = ((x_1, x_2, \ldots, x_d, z, x_{d+1}, \ldots, x_n), d')$.
6: **end function**

In addition, in the three different presented configurations, the marks determined on the children are always compatible by construction. Analogously, the same reasoning follows for the case where two pentatopes share a triangular face.

4.2 Stage 2: conformal refinement of all-P_{PP} meshes

In this section, we present a conformal refinement algorithm with a bounded number of generated similarity classes for meshes composed uniquely by elements of type P_{PP}. This algorithm determines the second stage of the refinement method for any unstructured mesh presented in Section 4.

The procedure presented in this section is stated in terms of a cycle composed of four steps, presented in Fig. 6. In Fig. 7 the templates for the bisection and setting of the marked edges of the children are presented. Given an element of type P_{PP}, Fig. 7(a), this element is split into two P_{AA} elements setting their marks using the templates presented in Figs. 7(b) and 7(c). After that, the type P_{AA}, Fig. 7(d), is bisected into two A_{AA} types applying the templates of Figs. 7(e) and 7(f). Following, an element of type A_{AA}, Fig. 7(g), is bisected into two A_{PP} using the templates presented in Figs. 7(h) and 7(i). Finally, from the type A_{PP}, Fig. 7(j), we obtain again two P_{PP} types applying the templates of Figs. 7(k) and 7(l).

We highlight that in order to apply the templates of Figure 7 we need to reorder the vertices of a given P_{PP} element to match the canonical representation of Figure 7(a). Similarly, the two children in Figures 7(b) and 7(c) have to be reordered to obtain the canonical P_{AA} in Figure 7(d) and then apply the corresponding templates. This node reordering has to be performed after each bisection to locate the marks in the canonical representation of the templated fathers. In addition, we highlight that in Figures 7(d), 7(b) and 7(c) the marked edges of level 2 are assigned on potential edges (see definition in Sec. 2). Although those edges do not exist on the parent, they exist in the children and grandchildren, where they will be used to determine the bisection edge.

Next, in Remark 5 we detail that this templated refinement procedure is analogous to Maubach's algorithm [11] when applied successively to one element.

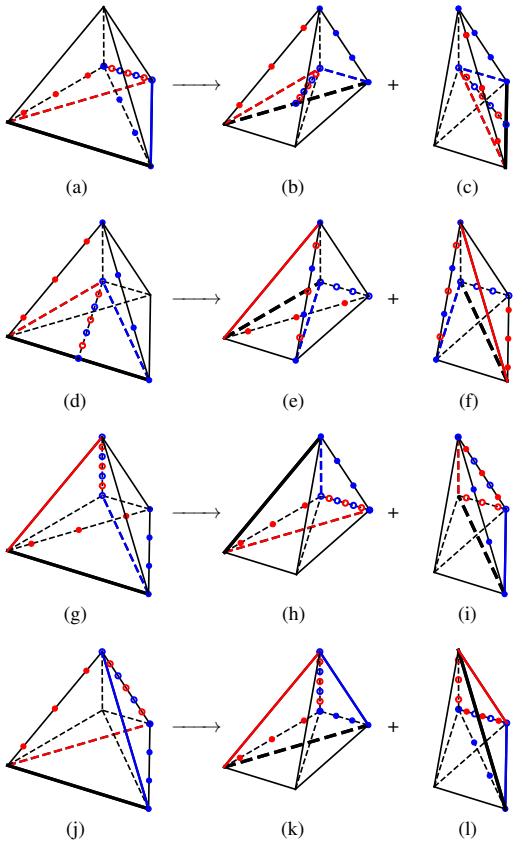


Fig. 7 Templates to perform the refinement cycle presented in Figure 6. (a)-(c) An element of type P_{PP} is bisected into two P_{AA}. (d)-(f) P_{AA} is bisected into two A_{AA}. (g)-(i) A_{AA} is bisected into two A_{PP}. (j)-(l) A_{PP} is bisected into two P_{PP}.

Maubach's algorithm cannot be applied in general to any given unstructured mesh as detailed in [11, 13]. Thus, finally in Remark 6, we analyze the conformity of the application of our approach for meshes composed of P_{PP} elements.

Remark 5 (Analogy to Maubach's algorithm). The refinement cycle in Fig. 6 performed using the templates presented in Fig. 7 is analogous to Maubach's algorithm [11] (see Alg. 3) when applied to a single pentatope. This analogy is interpreted as follows. Given a pentatope to bisect using Maubach's algorithm with a tag d, we consider as marked edge of level 0 the tagged edge. Next, we consider as marked edges of level 1 the tagged edges of the two children in the next application

Table 1 Permutations from the Maubach Algorithm 3 to canonical types in Figure 7

Canonical type	Tag in Algorithm 3	Permutation to obtain canonical representation
P_{PP}	$d=2$	$(0,2,1,3,4)$
P_{AA}	$d=1$	$(0,1,2,3,4)$
A_{AA}	$d=4$	$(0,4,2,3,1)$
A_{PP}	$d=3$	$(0,3,2,1,4)$

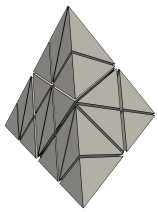


Fig. 8 Tetrahedral face of a pentatope of type P_{PP} after five refinements of the face .

of Maubach's algorithm. Analogously, we consider as marked edges of level 2 the tagged edges of the four grandchildren. Next, we find the permutation of the vertices $[x_0x_1x_2x_3x_4]$ to align the marks on the edges of the element with the canonical representation from Fig. 7. The obtained permutations are presented in Table 1.

Remark 6 (Conformal refinement for all-P_{PP} meshes). The refinement using Stage 2 of a marked mesh composed by elements of type P_{PP} leads to a conformal mesh. To illustrate the conformity of the refined mesh, we analyze two different cases:

- First, we analyze the case of the refinement of a single element. Since our method is analogous to Maubach's by Remark 5, it is also conformal when there is a single element successively refined, see details in [11, 13].
- Second, we analyze the conformity between the interface of adjacent elements of type P_{PP} with compatible marks. Extending the reasoning for tetrahedra in [13], it is sufficient to check if the bisection structure determined on a shared face is the same from both sides. Given a P_{PP} element to be refined, if we obtain the same refined mesh on all its tetrahedral faces we can ensure that the refinement of two adjacent P_{PP} is also conformal when using the `RefineToConformity` strategy. In particular, if we refine five times any of the five tetrahedral faces of a given P_{PP} the same tetrahedral mesh is obtained for all of them. This refined face mesh is illustrated in Figure 4.2 and is composed by 32 tetrahedra. The same reasoning follows for the case where two pentatopes share a triangular face.

Hence, if a pentatopic mesh can be marked with all elements as P_{PP}, then it can be conformingly refined using our analogy to Maubach's algorithm combined with the `RefineToConformity` strategy. This is the case when any given mesh is refined two times with Stage 1 in Sec. 4.1.

4.3 Properties of the method

In this section, we analyze the two main properties of the refinement procedure determined by Algorithm 2. Our refinement procedure requires as input a conformal unstructured 4D simplicial mesh. Given a set of elements to refine, the resulting mesh is a locally refined unstructured 4D simplicial mesh that is conformal and has a bounded number of generated similarity classes. These properties are discussed in the following remarks.

Remark 7 (Conformal refinement). The algorithm presented in Section 4 generates a conformal mesh. To show this, we take into account that this algorithm combines two refinement methods. The two first refinement steps in Stage 1 are performed by the algorithm presented in Sec. 4.1. After two refinements the elements are refined in Stage 2 according to the cycle in Fig. 6, see Sec. 4.2. In the worst case scenario, to prove conformal mesh refinement, all the elements of the initial mesh have to be twice refined at Stage 1. At this point, all the elements of the mesh are of type P_{PP} with compatible marks, as detailed in Remark 4. Then, the conformity of the refinement is ensured by Remark 6.

Remark 8 (Bounded number of generated similarity classes). The number of similarity classes produced by the repeated application of the cycle presented in Fig. 6 to an element is bounded by 1536. To prove this bound, we take into account that in the refinement scheme of Figure 6 it is required to perform two bisection steps before entering in the cycle. For each bisection, we generate at most two new similarity classes. Hence, from the given initial element, the bound of the similarity classes after the two first steps is $2 \cdot 2 = 4$. As highlighted in Remark 5 from Sec. 4.2, this second stage is analogous to Maubach's algorithm when applied to a single pentatope. In [13] it is proved that in 4D Maubach's algorithm has a sharp bound of 384 generated similarity classes for an element. Thus, the bound for the procedure of Figure 6 is $4 \cdot 384$, that is 1536.

5 Results

In this section, we present several results to illustrate the features and the applicability of the presented refinement scheme. In all the examples, we plot the minimum and maximum shape quality [16] in each refinement step of Alg. 1. To visualize the results we intersect each 4D mesh with a hyperplane to obtain a 3D cut that can be visualized. In Section 5.1, we refine an equilateral pentatope with two different initial marking configurations to illustrate that the similarity classes are bounded. In Section 5.2, we refine an unstructured 4D mesh to capture a hypersphere and, finally, in Section 5.3 we refine a simplicial mesh on a hypercube to capture a moving sphere.We highlight that in all the presented examples it has been explicitly checked that the generated meshes are conformal after the applied

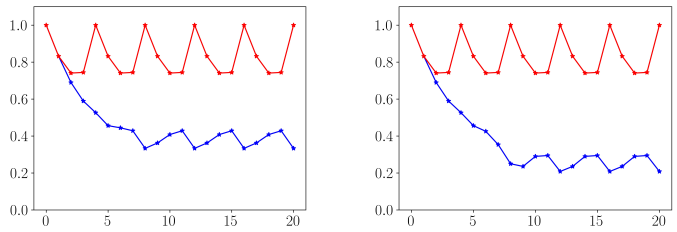


Fig. 9 Quality versus the number of iterations of the `RefineToConformity` algorithm applied to an equilateral pentatope marked as (a) P_{AA} and (b) A_{PO} type. The blue (red) line corresponds to the minimum (maximum) of the element shape quality at each iteration.

`RefineToConformity` strategy by checking that the only boundary faces of the mesh are on the boundary of the domain.

5.1 Bounded quality: iterative refinement of one pentatope

In this example, we check that our implementation of the refinement algorithm does not lead to degenerated elements after successive refinement of a given pentatope type. We enforce an equilateral pentatope to be marked as P_{AA} and a second equilateral pentatope to be marked as A_{PO}. Then, both pentatope types are globally refined 20 times. Figure 9 shows the minimum and maximum element quality at each refinement step. We observe that the minimum quality (vertical axis) decreases on the first refinement steps (horizontal axis) until a minimum value is reached. Then, the minimum and maximum qualities start to cycle every 4 refinement steps. This is an indicator of the bound of the number of generated similarity classes.

5.2 4D unstructured mesh: refining an extruded sphere octant

This example shows that the proposed refinement scheme can be applied to unstructured 4D pentatopic meshes. To this end, we generate an unstructured 4D mesh of a 3D sphere octant, of radius 1 and centered in the origin, extruded one unit along the fourth dimension. Then, we successively refine those elements that intersect a hypersphere of radius $1/4$ and centered in the origin. To generate the 4D mesh, we first generate an unstructured 3D mesh of the sphere octant composed by 40 nodes and 95 elements, see Figure 10(a). Then, we embed two copies of the 3D mesh points in the 4D space by setting the fourth coordinate to 0 and 1, respectively. Finally,

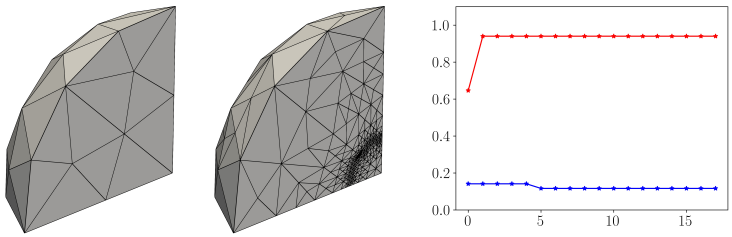


Fig. 10 A slice with the hyperplane $t = 0$ of the 4D simplicial mesh is illustrated for (a) the initial configuration and (b) after 17 iterations of `RefineToConformity`. (c) Minimum (blue) and maximum (red) element quality for each refinement step.

these 80 points are reconnected, using the implementation of the Delaunay algorithm provided by QHull [17], to obtain the unstructured 4D mesh. After applying 17 times the `RefineToConformity` algorithm, we obtain a 4D mesh composed by 8072909 elements and 433887 nodes. Figure 10(b), shows the tetrahedral mesh that corresponds to the boundary of the 4D pentatopic mesh at the base of the extrusion along the fourth dimension. Figure 10(c) shows the quality at each refinement step, where we can observe a lower quality bound is constant at value 0.11.

5.3 Space-time mesh: refining a sphere moving along the z-axis

Finally, we illustrate our application of interest, the refinement of a 4D mesh corresponding to a space-time representation, with varying resolution, of the temporal evolution of a 3D moving object. We consider a sphere of radius $1/5$ centered in the origin that moves along the z-axis from 0 to 1 with constant velocity 1. We generate an initial mesh on the hypercube $[0,1]^4$ composed by 24 pentatopes using Freudenthal-Kuhn algorithm [1–3]. Next, we apply 25 times the algorithm `RefineToConformity` to refine those elements that intersect the 4D sphere extrusion that represents the moving sphere. The final 4D mesh is composed by 5233296 pentatopes and 251457 nodes and it is illustrated in Figure 11. Figures 11(a)-11(c) show three slices of the mesh at $t = 0$, $t = 1/2$ and $t = 1$, respectively. We can observe that each one of the slices on t shows different positions of the moving sphere, from the initial point (0,0,0) at $t = 0$ to the final point $(0,0,1)$ at $t = 1$. In contrast with these three slices, in Figure 11(d) we show an slice of the mesh at $x = 0$. In the closest quadrilateral face of Fig. 11(d) we observe the path of the sphere on the surface of dimension 2 defined by the axis z and t at $x = y = 0$. In this quadrilateral face, we can see that the center of the sphere describes a straight line going from the lower left corner $(0,0,0,0)$ up to the top right corner $(0,0,1,1)$.

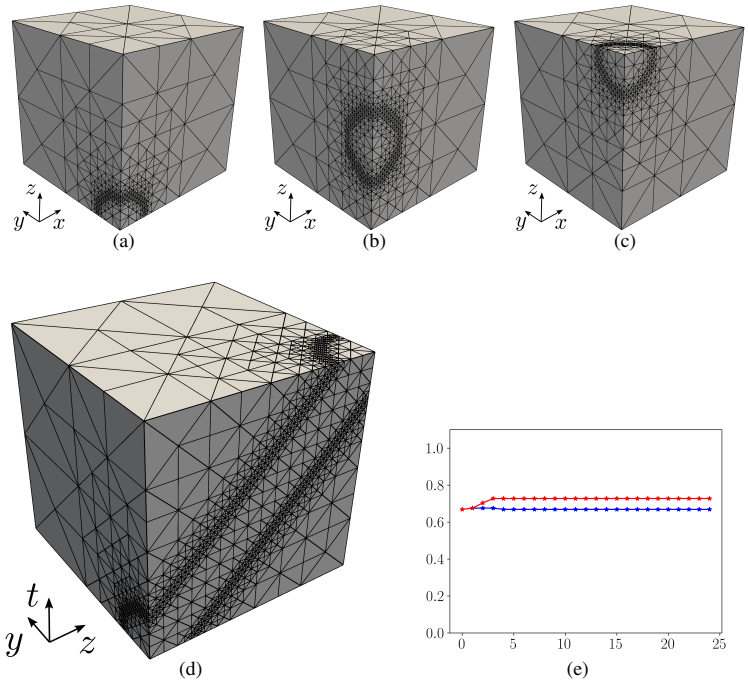


Fig. 11 Slice of the 4D simplicial mesh of the hypercube with the hyperplane: (a) $t = 0$, (b) $t = 0.5$, (c) $t = 1$ and (d) $x = 0$. (e) Minimum (blue) and maximum (red) element quality for each refinement step.

This is so since the sphere goes from $z = 0$ to $z = 1$ with constant velocity starting at $t = 0$ and finalizing at $t = 1$. Specifically, the location on the z-axis of the sphere is $z = t$.

6 Concluding remarks

In this work, we have presented a new refinement method via edge bisection for 4D pentatopic meshes. This method ensures that the mesh quality does not degenerate after successive refinements of a given element. To develop this method, we require to classify the elements of the mesh into different types in a similar fashion to [13]. Using the pentatope classification we provide four refinement templates to perform a cyclic bisection analogous to Maubach's method [11]. Combining two initializing refinements (Stage 1) with this templated refinement (Stage 2) we obtain a refine-

ment strategy that can be applied to any given pentatopic mesh. Using this method a finite number of similarity classes are generated when a given element is refined.

We apply the refinement scheme to different meshes to illustrate its features. First, we analyze that the mesh quality of the refinement of different element types does not degenerate. Second, we illustrate the applicability of the technique to refine unstructured 4D simplicial meshes. Finally, we analyze a space-time configuration of a sphere moving along an axis.

Acknowledgements This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715546. This work has also received funding from the Generalitat de Catalunya under grant number 2017 SGR 1731. The work of X. Roca has been partially supported by the Spanish Ministerio de Economía y Competitividad under the personal grant agreement RYC-2015-01633.

References

1. H Freudenthal. Simplizialzerlegungen von beschrankter flachheit. *Ann. Math.*, pages 580–582, 1942.
2. H Kuhn. Some combinatorial lemmas in topology. *IBM Journal of research and development*, 4(5):518–524, 1960.
3. J Bey. Simplicial grid refinement: on freudenthal's algorithm and the optimal number of congruence classes. *Numerische Mathematik*, 85(1):1–29, 2000.
4. R Bank, A Sherman, and A Weiser. Some refinement algorithms and data structures for regular local mesh refinement. *Scientific Computing*, 1:3–17, 1983.
5. J Bey. Tetrahedral grid refinement. *Computing*, 55(4):355–378, 1995.
6. A Liu and B Joe. Quality local refinement of tetrahedral meshes based on 8-subtetrahedron subdivision. *Mathematics of Computation*, 65(215):1183–1200, 1996.
7. S Zhang. Successive subdivisions of tetrahedra and multigrid methods on tetrahedral meshes. *Houston J. Math*, 21(3):541–556, 1995.
8. MC Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International journal for numerical methods in Engineering*, 20(4):745–756, 1984.
9. E Bänsch. Local mesh refinement in 2 and 3 dimensions. *IMPACT of Computing in Science and Engineering*, 3(3):181–191, 1991.
10. A Liu and B Joe. Quality local refinement of tetrahedral meshes based on bisection. *SIAM Journal on Scientific Computing*, 16(6):1269–1291, 1995.
11. J Maubach. Local bisection refinement for n-simplicial grids generated by reflection. *SIAM Journal on Scientific Computing*, 16(1):210–227, 1995.
12. C Traxler. An algorithm for adaptive mesh refinement in *n* dimensions. *Computing*, 59(2):115–137, 1997.
13. D Arnold, A Mukherjee, and L Pouly. Locally adapted tetrahedral meshes using bisection. *SIAM Journal on Scientific Computing*, 22(2):431–448, 2000.
14. A Plaza and MC Rivara. Mesh refinement based on the 8-tetrahedra longest-edge partition. In *IMR*, pages 67–78, 2003.
15. M Neumüller and O Steinbach. A flexible space-time discontinuous galerkin method for parabolic initial boundary value problems. *Berichte aus dem Institut für Numerische Mathematik*, 2, 2011.
16. P. M. Knupp. Algebraic mesh quality metrics. *SIAM J. Numer. Anal.*, 23(1):193–218, 2001.
17. C Barber, D Dobkin, and H Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

Bibliography

Alkämper, M., F. Gaspoz, and R. Klöfkorn (2018). A weak compatibility condition for newest vertex bisection in any dimension. *SIAM Journal on Scientific Computing 40*(6), A3853–A3872.

Arnold, D. N., A. Mukherjee, and L. Pouly (2000). Locally adapted tetrahedral meshes using bisection. *SIAM Journal on Scientific Computing 22*(2), 431–448.

Bänsch, E. (1991). Local mesh refinement in 2 and 3 dimensions. *IMPACT of Computing in Science and Engineering 3*(3), 181–191.

Belda-Ferrín, G., A. Gargallo-Peiró, and X. Roca (2019). Local Bisection for Conformal Refinement of Unstructured 4D Simplicial Meshes. In *27th International Meshing Roundtable*, Volume 127, pp. 229–247. Springer International Publishing.

Belda-Ferrín, G., E. Ruiz-Gironés, A. Gargallo-Peiró, and X. Roca (2019). Visualization of pentatopic meshes. Technical report, 28th International Meshing Roundtable.

Belda-Ferrín, G., E. Ruiz-Gironés, A. Gargallo-Peiró, and X. Roca (2022). Marked bisection for local refinement of n-dimensional unstructured conformal meshes. *Computer-Aided Design*. Submitted on April 22, 2022.

Belda-Ferrín, G., E. Ruiz-Gironés, and X. Roca (2021). Bisecting with optimal similarity bound on 3D unstructured conformal meshes. In *2022 SIAM International Meshing Roundtable (IMR), Virtual Conference*. Zenodo.

Belda-Ferrín, G., E. Ruiz-Gironés, and X. Roca (2022a). Newest vertex bisection for unstructured n-simplicial meshes. Paper in preparation.

Belda-Ferrín, G., E. Ruiz-Gironés, and X. Roca (2022b). Suitability of marked bisection for local refinement of n-dimensional unstructured conformal meshes. Paper in preparation.

Binev, P., W. Dahmen, and R. DeVore (2004). Adaptive finite element methods with convergence rates. *Numerische Mathematik 97*, 219–268.

Caplan, P. C. (2019). *Four-Dimensional Anisotropic Mesh Adaptation for Spacetime Numerical Simulations*. Ph. D. thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology.

Coxeter, H. S. (1934). Discrete groups generated by reflections. *Annals of Mathematics 35*(3), 588–621.

Freudenthal, H. (1942). Simplizialzerlegungen von beschrankter flachheit. *Annals of Mathematics 43*(3), 580–582.

Knupp, P. M. (2001). Algebraic mesh quality metrics. *SIAM Journal on Scientific Computing 23*(1), 193–218.

Kossaczký, I. (1994). A recursive approach to local mesh refinement in two and three dimensions. *Journal of Computational and Applied Mathematics 55*(3), 275–288.

Kuhn, H. W. (1960). Some combinatorial lemmas in topology. *IBM Journal of Research and Development 4*(5), 518–524.

Liu, A. and B. Joe (1994). On the shape of tetrahedra from bisection. *Mathematics of Computation 63*(207), 141–154.

Liu, A. and B. Joe (1995). Quality local refinement of tetrahedral meshes based on bisection. *SIAM Journal on Scientific Computing 16*(6), 1269–1291.

Maubach, J. M. (1995). Local bisection refinement for n-simplicial grids generated by reflection. *SIAM Journal on Scientific Computing 16*(1), 210–227.

Maubach, J. M. (1996a). The amount of similarity classes created by local n-simplicial bisection refinement. *preprint*.

Maubach, J. M. (1996b). The efficient location of neighbors for locally refined n-simplicial grids. *5th International Meshing Roundable 4*(6), 137–153.

Mitchell, W. F. (1991). Adaptive refinement for arbitrary finite-element spaces with hierarchical bases. *Journal of Computational and Applied Mathematics 36*(1), 65–78. Special Issue on Adaptive Methods.

Mitchell, W. F. (2017). 30 years of newest vertex bisection. *Journal of Numerical Analysis, Industrial and Applied Mathematics 11*(1), 11–22.

Neumüller, M. and E. Karabelas (2019). Generating admissible space-time meshes for moving domains in $(d + 1)$ dimensions. In *Space-Time Methods: Applications to Partial Differential Equations*, Chapter 6, pp. 185–206. De Gruyter.

Neumüller, M. and O. Steinbach (2011). Refinement of flexible space–time finite element meshes and discontinuous galerkin methods. *Computing and Visualization in Science 14*(5), 189–205.

Persson, P.-O. and G. Strang (2004). A simple mesh generator in matlab. *SIAM Review 46*(2), 329–345.

Plaza, A. and G. F. Carey (2000). Local refinement of simplicial grids based on the skeleton. *Applied Numerical Mathematics 32*(2), 195–218.

Plaza, A. and M.-C. Rivara (2003). Mesh refinement based on the 8-tetrahedra longest-edge partition. In *Proceedings of the 12th International Meshing Roundtable*, pp. 67–78.

Rivara, M.-C. (1984). Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering 20*(4), 745–756.

Rivara, M.-C. (1991). Local modification of meshes for adaptive and/or multigrid finite-element methods. *Journal of Computational and Applied Mathematics 36*(1), 79–89. Special Issue on Adaptive Methods.

Stevenson, R. (2008). The completion of locally refined simplicial partitions created by bisection. *Mathematics of Computation 77*(261), 227–241.

Traxler, C. T. (1997). An algorithm for adaptive mesh refinement in n dimensions. *Computing 59*(2), 115–137.