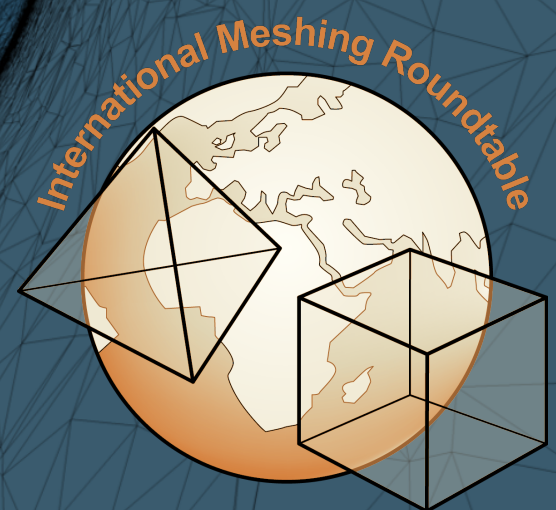


# Proceedings of the 2022 SIAM International Meshing Roundtable



**22-25 Feb, 2022  
Virtual Conference**



**Proceedings of the 2022 SIAM International  
Meshing Roundtable**

Trevor Robinson

David Moxey

Vladimir Z. Tomov



2022 SIAM International Meshing Roundtable  
Virtual Conference, February 22-25, 2022

*Editors:*

Trevor Robinson, Queen's University Belfast

David Moxey, King's College London

Vladimir Z. Tomov, Lawrence Livermore National Laboratory\*



<https://internationalmeshingroundtable.com>

Cover art adapted from the winning “Meshing Maestro” entry by  
Mike Park, Ray Gomez, and Bil Kleb of NASA.

Space Needle model from <https://grabcad.com/library/seattle-space-needle-2>

\*Performed under the auspices of the U.S. Department of Energy under Contract  
DE-AC52-07NA27344 (LLNL-PROC-835065)

Copyright ©2022 held by the authors of the individual papers.

Distribution of the material in this volume is permitted under the Creative Commons Attribution 4.0 International License.

ISBN: 978-1-7334890-2-7

DOI: 10.5281/zenodo.6562462



## Contents

High-Order Metric Interpolation for Curved $R$ -Adaption by Distortion Minimization	
<i>Guillermo Aparicio-Estrems, Abel Gargallo-Peiró, and Xevi Roca</i>	1
An Efficient Solver to Approximate CAD Curves with Super-Convergent Rates	
<i>Julia Docampo-Sánchez, Eloi Ruiz-Gironés, and Xevi Roca</i>	13
Automatic Generation of Load-Balancing-Aware Block-Structured Grids for Complex Ocean Domains	
<i>Daniel Zint, Roberto Grosso, Vadym Aizinger, Sara Faghih-Naini, Sebastian Kuckuk, and Harald Köstler</i>	25
Local Decomposition of Hexahedral Singular Nodes into Singular Curves	
<i>Paul Zhang, Judy (Hsin-Hui) Chiang, Xinyi (Cynthia) Fan, and Klara Mundilova</i>	37
Incremental Decomposition for Hex-Meshing in CAD Using Virtual Topology	
<i>Benoit Lecallard, Trevor T. Robinson, Cecil G. Armstrong, Declan C. Nolan, and Harsha Ramesh</i>	49
Interactive Visualization of Large and Arbitrary Polygonal and Polyhedral Meshes with OpenGL 4	
<i>Matthieu Maunoury, Rémi Feuillet, and Adrien Loseille</i>	61
$P^3$ Bézier CAD Surrogates for Anisotropic Mesh Adaptation	
<i>Adrien Loseille and Lucien Rochery</i>	74
Bisecting with Optimal Similarity Bound on 3D Unstructured Conformal Meshes	
<i>Guillem Belda-Ferrín, Eloi Ruiz-Gironés, and Xevi Roca</i>	86
Parallel Four-Dimensional Anisotropic Mesh Adaptation	
<i>Philip Claude Caplan</i>	98
Shrink Wrap Mesh Generation Using Morphological Operators with Selected Applications	
<i>Vijai Kumar Suriyababu, Cornelis Vuik, and Matthias Möller</i>	110
Smoothing of Shell Meshes on Faceted B-rep Geometry	
<i>Harold J. Fogg and Jonathan E. Makem</i>	130







## Preface

The papers in this volume were peer-reviewed and selected for presentation at the SIAM International Meshing Roundtable Workshop (IMR), held Feb 22-25, 2022. The conference was originally planned to take place in Seattle, USA, but was ultimately run as a virtual conference. This was the first IMR in collaboration with the Society for Industrial and Applied Mathematics (SIAM).

The International Meshing Roundtable was started by Sandia National Laboratories in 1992 as a small meeting of organizations striving to establish a common focus for research and development in the field of mesh generation. Now after 30 years, it has become clear that the International Meshing Roundtable has become the recognized international focal point for state-of-the-art meshing research collaboration spanning research and development from universities, commercial companies, and government laboratories.

The SIAM International Meshing Roundtable 2022 consisted of presentations of peer-reviewed technical papers, research notes, keynote and invited talks, short course presentations, a poster session and competition, and a meshing contest. This year we have made the proceedings openly accessible by hosting them on Zenodo under a Creative Commons license. The Program Committee would like to express our appreciation to all who participate in making the International Meshing Roundtable a successful and enriching experience.

The papers in these proceedings present novel contributions that range from the theoretical to practical. This year, the committee selected 11 papers based on the input from peer reviewers. Reviewers assessed the papers based on quality, originality, and appropriateness to the theme of the International Meshing Roundtable. We would like to thank all who submitted and participated in the IMR. We also extend our appreciation to the colleagues who provided reviews of the submitted papers. Their efforts were essential to the process of selecting papers for the IMR. The names of the reviewers are acknowledged in the following pages.

We thank Siemens for providing the Microsoft Teams platform that was used to manage the virtual conference, and to our awards sponsors Cadence and Design-FOIL. We deeply acknowledge the support of the IMR Steering Committee, whose primary goal is to determine the future directions of the International Meshing Roundtable. The steering committee members are: Scott Canann (Siemens), Steve Karman (Oak Ridge National Laboratory), Trevor Robinson (Queen's University Belfast), Suzanne Shontz (University of Kansas), and John Verdicchio (Siemens). We extend special thanks to Kathy Loeppky of Sandia National Laboratories for her time and effort to make the SIAM IMR 2022 a success.

May 2022,  
SIAM IMR 2022 Program Committee





## List of Reviewers

David Bommes	Navamita Ray
Jean Cabello	Xevi Roca
Marcel Campen	Sergio Salinas
Philip Caplan	Robert Schneiders
Jean-Christophe Cuilliere	Suzanne Shontz
Julia Docampo-Sánchez	Hang Si
Daming Feng	Dmitry Sokolov
Nicola Ferro	John Steinbrenner
Harry Fogg	José Pablo Suárez Rivero
W. Randolph Franklin	Vijai Kumar Suriyababu
Rao Garimella	John Verdicchio
Nancy Hitschfeld	Nicholas Vining
Franck Ledoux	Shawn Walker
Weiyang Lin	Rui Wang
Adrien Loseille	Jean-Christophe Weill
Ahmed Mahmoud	Soji Yamakawa
Loïc Maréchal	Jessica Zhang
Matthieu Maunoury	Xiaopeng Zheng
David McLaurin	Daniel Zint
Erik Melin	
Scott Mitchell	
Ketan Mittal	
Walter Nissen	
Mike Park	
Joaquim Peiró	
Alexander Rand	





## **Committee Members**

Trevor Robinson - Queen's University Belfast - Committee Chair

David Moxey - King's College London - Papers Chair

Vladimir Z. Tomov - Lawrence Livermore National Laboratory - Papers Chair

Eloi Ruiz-Gironés - Barcelona Supercomputing Center

Na Lei - Dalian University of Technology

Xianfeng Gu - Stony Brook University

Chaman Singh Verma - Avail MedSystems

Julian Marcon - NASA Ames Research Center

Jonathan Makem - Siemens PLM Software

Carolyn Woeber - Cadence

Braxton Osting - University of Utah

# HIGH-ORDER METRIC INTERPOLATION FOR CURVED $R$ -ADAPTION BY DISTORTION MINIMIZATION

Guillermo Aparicio-Estrems<sup>1</sup>

Abel Gargallo-Peiró<sup>2</sup>

Xevi Roca<sup>3</sup>

<sup>1</sup>*Barcelona Supercomputing Center, 08034 Barcelona, Spain guillermo.aparicio@bsc.es*

<sup>2</sup>*Barcelona Supercomputing Center, 08034 Barcelona, Spain abel.gargallo@bsc.es*

<sup>3</sup>*Barcelona Supercomputing Center, 08034 Barcelona, Spain xevi.roca@bsc.es*

## ABSTRACT

We detail how to use Newton’s method for distortion-based curved  $r$ -adaption to a discrete high-order metric field. To this end, we consider three existent ingredients. First, a specific-purpose solver for distortion minimization. Second, a log-Euclidean high-order metric interpolation. Third, a point localization procedure for curved high-order meshes. We also extend to discrete metric fields a distortion-based curved  $r$ -adaption framework. To extend the framework, we provide, for the log-Euclidean high-order metric interpolation, the first and second derivatives in physical coordinates. These derivatives are required by Newton’s method to solve the distortion minimization. The distortion minimization allows properly matching the anisotropic curved features of a discrete high-order metric. This matching capability might be relevant in global and cavity-based curved (straight-edged) high-order mesh adaption.

**Keywords:** Anisotropy,  $r$ -adaption, metric interpolation, curved high-order meshes

## 1. INTRODUCTION

The capability to relocate mesh nodes without changing the mesh topology, referred to as  $r$ -adaptivity, is a key ingredient in many adaptive PDE-based applications [1–3]. In these applications, to improve the solution accuracy, an error indicator or estimator determines the target stretching and alignment of the mesh. Then, to match these target features, an  $r$ -adaption procedure modifies the whole mesh (global) [4,5] or a previously remeshed cavity (local) [6–8].

In either case,  $r$ -adaptivity contributes to increasing the solution accuracy for a fixed number of degrees of freedom supported on a straight-edged mesh [3,4,6,9,10]. However, straight-edged meshes might not be an efficient support in many applications. Especially in applications where additional straight-edged mesh elements are artificially required to match highly curved solution features [11].

To efficiently match curved features, many practitioners have recently started to exploit curved high-order meshes. These meshes can be stretched and aligned in a pointwise varying fashion through anisotropic procedures [12], geodesic

approaches for curved edges [13,14], shock-tracking methods [15–17], and deformation analogies [18,19]. Alternatively, the curved  $r$ -adaption can be driven, as for straight-edged elements [4,5], by distortion measures. These measures are defined point-wise and are aware of either a target deformation matrix [20] or a target metric [21].

In adaptivity applications, the target deformations and metrics are not known a priori. These target fields are reconstructed a posteriori from the solution on the last mesh. Specifically, this mesh supports the resulting discrete representation of the target field. This discrete representation is key to interpolate the required field values in the adaptive procedure. Hence, to enable high-order adaptivity, we need the capability to interpolate target fields on a high-order mesh.

To match a deformation matrix, distortion optimization for curved  $r$ -adaption to a discrete target field is detailed in [20]. The method is really well-suited for simulation-driven  $r$ -adaption [22,23]. It evaluates the distortion in a physical point by interpolating the target matrix on a discrete field. Although the derivatives of the target matrices are not zero,

the method assumes they are zero. Moreover, the second derivatives are also assumed to be zero. Since non-null derivatives are assumed to be zero although the approach implements Newton’s method, the curved  $r$ -adaption minimization corresponds to a quasi-Newton method.

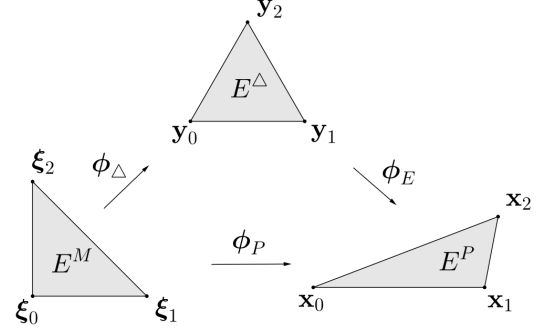
To match a metric, distortion-based curved  $r$ -adaption to an analytic field can be performed with Newton’s minimization [21, 24, 25]. The formulation for an analytic metric is derived in [21], while a specific-purpose globalization and a pre-conditioned Netwon-CG method are proposed in [24, 25] to minimize the mesh distortion. Since the method deals with an analytic metric, it does not specify the derivatives for a metric represented by a discrete high-order field.

Regarding a discrete field representation, a convenient approach is to use a log-Euclidean [26] high-order metric interpolation [27]. This metric interpolation drives a cavity-based adaption approach, where the remeshed cavities are improved by locally smoothing the curved quadratic edges. To smooth these edges, the method optimizes the mid-node position. The optimization only uses the first derivatives of the log-Euclidean metric interpolation in terms of the curved edge coordinates. Accordingly, the method does not provide the first and second derivatives of the discrete metric field in physical coordinates.

Considering the previous open issues, our main contribution is to use Newton’s optimization for distortion-based curved  $r$ -adaption to a discrete high-order metric field. We need three existent ingredients. First, to minimize the distortion, we use the specific-purpose solver in [24, 25]. Second, we represent the metric field as a log-Euclidean high-order metric interpolation [27] on a curved high-order mesh. Third, we locate physical points in the curved background mesh similar to the approach in [22]. We also need to extend to discrete metric fields a distortion-based curved  $r$ -adaption framework [21].

To extend the framework, the main novelty is to provide, for the log-Euclidean high-order metric interpolation, the first and second derivatives in physical coordinates. These derivatives are critical to use Newton’s method for distortion minimization. This minimization leads to unprecedented second-order optimization results for curved  $r$ -adaption for a discrete high-order metric representation on a curved (or straight-edged) mesh.

The remainder of this paper is organized as follows. First, in Section 2 we introduce the metric-aware measures for curved high-order 2D elements. Next, in Section 3 we introduce the high-order log-Euclidean metric interpolation framework and we present the computation of its gradient and Hessian. Following, we present several examples to illustrate the capabilities of the proposed framework, Section 4. To finalize, in Section 5 we present the main conclusions and sum up the future work to develop.



**Figure 1:** Mappings between the master, the ideal, and the physical elements in the linear case.

## 2. PRELIMINARIES: METRIC-AWARE MEASURES FOR CURVED HIGH-ORDER ELEMENTS

In this section, we review the definition of the Jacobian-based quality measure for high-order elements equipped with a metric, presented in [21]. To define and compute a Jacobian-based measure for simplices [5], three elements are required: the master, the ideal, and the physical, see Figure 1 for the linear triangle case. The master ( $E^M$ ) is the element from which the iso-parametric mapping is defined. The equilateral element ( $E^\Delta$ ) represents the target configuration in the isotropic case. The physical ( $E^P$ ) is the element to be measured.

To summarize the results in [21], we present the expression of the metric distortion measure in terms of the equilateral element  $E^\Delta$ . First, we need to compute a mapping from the master to the equilateral and physical elements, denoted as  $\phi_\Delta$  and  $\phi_P$ , respectively. By means of these mappings, we determine a mapping between the equilateral and physical elements by the composition

$$\phi_E : E^\Delta \xrightarrow{\phi_\Delta^{-1}} E^M \xrightarrow{\phi_P} E^P.$$

As detailed in [21], we define the point-wise distortion measure for a high-order element  $E^P$  equipped with a point-wise metric  $\mathbf{M}$ , at a point  $\mathbf{y} \in E^\Delta$  as

$$\mathcal{N}\phi_E(\mathbf{y}) = \frac{\text{tr}(\mathbf{D}\phi_E(\mathbf{y})^T \cdot \mathbf{M}(\phi_E(\mathbf{y})) \cdot \mathbf{D}\phi_E(\mathbf{y}))}{d \left( \det(\mathbf{D}\phi_E(\mathbf{y})^T \cdot \mathbf{M}(\phi_E(\mathbf{y})) \cdot \mathbf{D}\phi_E(\mathbf{y})) \right)^{1/d}}, \quad (1)$$

where the Jacobian of the map  $\phi_E$  is given by

$$\mathbf{D}\phi_E(\mathbf{y}) := \mathbf{D}\phi_P(\phi_\Delta^{-1}(\mathbf{y})) \cdot \mathbf{D}\phi_\Delta^{-1}(\mathbf{y}).$$

Note that the distortion measure is independent of the computation of the metric  $\mathbf{M}(\phi_E(\mathbf{y}))$ , either using an analytical or a discretized representation.

We regularize the determinant in the denominator of Eq. (1) in order to detect inverted elements [28–31]. In particular,



we define

$$\sigma_0 = \frac{1}{2}(\sigma + |\sigma|),$$

where

$$\sigma = \det(\mathbf{D}\phi_E(\mathbf{y})) \sqrt{\det(\mathbf{M}(\phi_E(\mathbf{y})))}.$$

Then, we define the point-wise regularized distortion measure of a physical element  $E^P$  at a point  $\mathbf{y} \in E^\Delta$  as

$$\mathcal{N}_0\phi_E(\mathbf{y}) := \frac{\text{tr}(\mathbf{D}\phi_E(\mathbf{y})^\text{T} \cdot \mathbf{M}(\phi_E(\mathbf{y})) \cdot \mathbf{D}\phi_E(\mathbf{y}))}{d\sigma_0^{2/d}}, \quad (2)$$

and its corresponding point-wise quality measure

$$\mathcal{Q}\phi_E(\mathbf{y}) = \frac{1}{\mathcal{N}_0\phi_E(\mathbf{y})}. \quad (3)$$

Finally, we define the regularized elemental distortion by

$$\eta_{(E^P, \mathbf{M})} := \frac{\left( \int_{E^\Delta} (\mathcal{N}_0\phi_E(\mathbf{y}))^2 d\mathbf{y} \right)^{1/2}}{(\int_{E^\Delta} 1 d\mathbf{y})^{1/2}} \quad (4)$$

and its corresponding quality

$$\mathfrak{q}_{(E^P, \mathbf{M})} = \frac{1}{\eta_{(E^P, \mathbf{M})}}. \quad (5)$$

We can improve the mesh configuration by means of relocating the nodes of the mesh according to a given distortion measure [21, 24, 25, 32]. In [21] it is proposed an optimization of the distortion (quality) of a mesh  $\mathcal{M}$  equipped with a target metric  $\mathbf{M}$  that describes the desired alignment and stretching of the mesh elements. To optimize a given mesh  $\mathcal{M}$ , first it is defined the mesh distortion by

$$\mathcal{F}(\mathcal{M}) := \sum_{E^P \in \mathcal{M}} \int_{E^\Delta} (\mathcal{N}_0\phi_E(\mathbf{y}))^2 d\mathbf{y},$$

which allows to pose the following global minimization problem

$$\mathcal{M}^* := \underset{\mathcal{M}}{\text{argmin}} \mathcal{F}(\mathcal{M}), \quad (6)$$

to improve the mesh configuration according to  $\mathcal{F}$ . In particular, herein, the degrees of freedom of the minimization problem in Eq. (6) correspond to the spatial coordinates of the mesh nodes.

To evaluate the distortion minimization formulation presented in Equation (6), an input metric is required. The reviewed r-adaption procedure has been applied for analytic metrics in [21]. In the following section, we detail the interpolation process that is required to extend the presented framework to discrete metrics.

### 3. LOG-EUCLIDEAN METRIC INTERPOLATION

In this section, we formulate a metric interpolation process that allows both the distortion evaluation, Eq. (2), and its optimization, Eq. (6). In Sec. 3.1 we detail the log-Euclidean

metric interpolation for linear and high-order elements first presented in [26] and [27, 33], respectively. Then, in Sec. 3.2 we present, as a contribution of this work, the gradient and the Hessian of the log-Euclidean interpolation. Their computation will be used for the distortion minimization problem.

#### 3.1 Metric Interpolation

In this section, we introduce the definition of the log-Euclidean metric interpolation at the background mesh. First, we introduce the required notation of the mappings and their parameters with the corresponding diagram. Secondly, we detail the interpolation procedure.

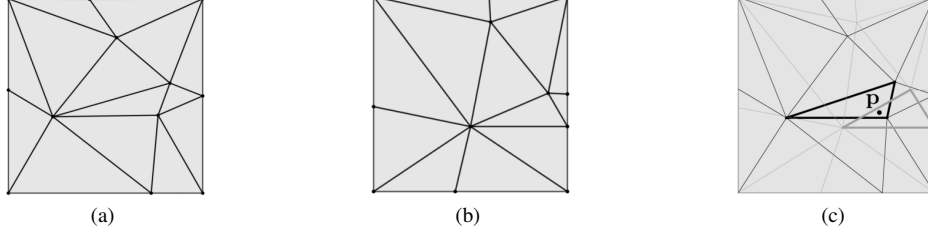
To evaluate the metric-aware distortion measure in Eq. (2) featuring discrete metrics, two meshes are required. On the one hand, the *physical* mesh  $\mathcal{M}$ , Figure 2(a), is the domain where the elements are deformed in order to solve the problem presented in Equation (6). On the other hand, the background mesh  $\hat{\mathcal{M}}$ , Figure 2(b), is a mesh that stores discrete metric values as a nodal field.

To evaluate the point-wise metric-aware distortion measure, we need to compute the interpolation of the point-wise metric values. For this, the localization between both meshes is required [22, 34, 35]. In particular, a physical point  $\mathbf{p} \in \mathcal{M}$  is located at the background mesh  $\hat{\mathcal{M}}$  where the metric is interpolated, see Figure 2(c). In what follows, we introduce the elements and the mappings required for this localization procedure.

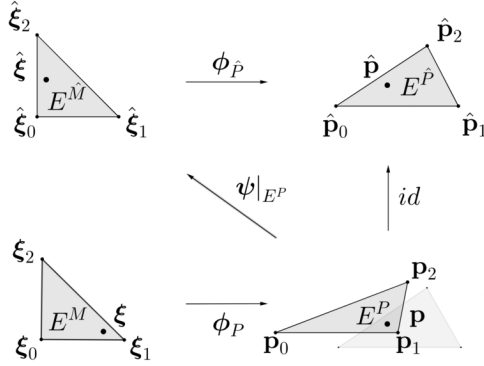
We integrate the distortion measure presented in Equation (2) over the equilateral element via the master element  $E^M$ . In particular, for the metric evaluation, we map via  $\phi_P$ , each integration point  $\xi \in E^M$  to a point  $\mathbf{p}$  of the physical element  $E^P$ , see Figure 3. To compute the metric at  $\mathbf{p}$  we need to locate  $\mathbf{p}$  in the background mesh, where the values of the metric are stored, see the intersection between  $E^P$  and the background element  $E^{\hat{P}}$  in Figure 3. In addition, Figure 3 shows the procedure to obtain the coordinate to interpolate the metric from the quadrature points. In particular, we map a reference point  $\xi \in E^M$  to a physical point  $\mathbf{p} = \phi_P(\xi) \in E^P$ , which we identify it with a point  $\hat{\mathbf{p}} \in E^{\hat{P}}$  of the background mesh and its preimage is the background reference point  $\hat{\xi} = \phi_{\hat{P}}^{-1}(\hat{\mathbf{p}}) \in E^{\hat{M}}$ .

Given a physical point  $\mathbf{p}$ , we find it convenient to denote by  $\psi$  any mapping from a background element containing  $\mathbf{p}$  that provides the coordinates in the background master element  $E^{\hat{M}}$ . Using this notation, we understand that any projection of a physical point  $\mathbf{p}$  onto a point  $\hat{\xi}$  of the background master element  $E^{\hat{M}}$  corresponds to the evaluation of the non-linear function  $\hat{\xi} = \psi(\mathbf{p})$ .

To evaluate this non-linear function, we exploit that the expression of  $\psi|_{E^P}$ , defined in the intersection of a physical



**Figure 2:** Point localization: (a) physical mesh, (b) background mesh, and (c) a point  $\mathbf{p}$  in the corresponding physical and background element (bold edges).



**Figure 3:** Mappings between the master and the physical elements (below) and their background analogs (above).

element  $E^P$  and a fixed background element  $E^{\hat{P}}$ , is given by

$$\begin{aligned} \psi|_{E^P} : E^P \cap E^{\hat{P}} &\rightarrow E^{\hat{M}} \\ \mathbf{p} &\mapsto \phi_{\hat{P}}^{-1}(\mathbf{p}). \end{aligned} \quad (7)$$

Specifically, we solve the non-linear inverse expression in the image term, Equation (7), by applying Newton's minimization to the squared distance. That is, we solve

$$\hat{\xi} = \underset{\xi}{\operatorname{argmin}} \left\| \phi_{\hat{P}}(\xi) - \mathbf{p} \right\|^2.$$

The result is a numerical approximation of the point coordinates in the background master element. An alternative approach [22] is to seek the zeros of the vector equation

$$\phi_{\hat{P}}(\hat{\xi}) - \mathbf{p} = \mathbf{0}.$$

Once the background master coordinates associated to a given physical point have been computed, it is necessary to interpolate the metric supported by the background mesh at the corresponding master coordinate. To do so, we use the log-Euclidean interpolation proposed in [26, 27]:

$$\mathbf{M}(\hat{\mathbf{N}}) := \exp(\mathbf{L}(\hat{\mathbf{N}})), \quad \mathbf{L}(\hat{\mathbf{N}}) := \sum_{j=1}^{\hat{n}} \hat{N}_j \log \mathbf{M}_j, \quad (8)$$

where for the  $j$ -th node of the master element  $E^{\hat{M}}$ ,  $\mathbf{M}_j$  and  $\hat{N}_j$  are the corresponding metric value and shape function, respectively. In addition,  $\hat{\mathbf{N}}$  denotes all the shape functions,  $\hat{n} = \frac{(\hat{p}+1)(\hat{p}+2)}{2}$  is the number of nodes and where  $\hat{p}$  is the interpolation degree which corresponds to the polynomial degree of the master element  $E^{\hat{M}}$ . Finally,  $\mathbf{M}(\hat{\mathbf{N}})$  is characterized by the *eigenvalue-based* matrix exponential function

$$\mathbf{M}(\hat{\mathbf{N}}) = \mathbf{U} \cdot \exp \mathbf{D} \cdot \mathbf{U}^T, \quad (9)$$

where  $\mathbf{D}$ ,  $\mathbf{U}$  are given from the eigenvalue decomposition of the matrix  $\mathbf{L}(\hat{\mathbf{N}}) =: \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{U}^T$ . Finally, for each physical point  $\mathbf{p}$  the metric interpolation is given by  $\mathbf{M}(\hat{\mathbf{N}}(\psi(\mathbf{p})))$ .

### 3.2 Gradient and Hessian

This section gives formulas of the gradient and Hessian of the metric interpolation over a background mesh in terms of the physical coordinates. For this, we detail first the case for the metric interpolation at a single element and then for the background mesh. In particular, our approach uses the gradient and Hessian of the eigenvalue decomposition presented in [36].

To compute the derivatives of the metric  $\mathbf{M}$  we first differentiate the eigenvalue-based exponential matrix function presented in Equation (9) and then we differentiate the  $\mathbf{L}$  function presented in Equation (8). By denoting  $x_j$  the coordinates of  $\mathbf{p}$  and  $\partial_j := \frac{\partial}{\partial x_j}$ ,  $\partial_{jk} := \partial_j \partial_k = \frac{\partial}{\partial x_j} \frac{\partial}{\partial x_k}$  the partial derivatives in terms of the physical coordinates of  $\mathbf{p}$ , we can compute the spatial derivatives of the metric interpolation of Equation (8). In particular, the first-order derivatives are given by

$$\begin{aligned} \partial_j \mathbf{M}(\hat{\mathbf{N}}) &= \partial_j \exp \mathbf{L}(\hat{\mathbf{N}}) = \partial_j (\mathbf{U} \cdot \exp \mathbf{D} \cdot \mathbf{U}^T) = \\ &= (\partial_j \mathbf{U}) \cdot \exp \mathbf{D} \cdot \mathbf{U}^T + \mathbf{U} \cdot (\partial_j \exp \mathbf{D}) \cdot \mathbf{U}^T + \\ &\quad \mathbf{U} \cdot \exp \mathbf{D} \cdot (\partial_j \mathbf{U}^T), \end{aligned}$$

and the second-order derivatives are given by

$$\begin{aligned}\partial_{jk}\mathbf{M}(\hat{\mathbf{N}}) &= \partial_{jk} \exp \mathbf{L}(\hat{\mathbf{N}}) = \partial_{jk} \left( \mathbf{U} \cdot \exp \mathbf{D} \cdot \mathbf{U}^T \right) = \\ &= (\partial_{jk} \mathbf{U}) \cdot \exp \mathbf{D} \cdot \mathbf{U}^T + \partial_k \mathbf{U} \cdot (\partial_j \exp \mathbf{D}) \cdot \mathbf{U}^T + \\ &+ \partial_k \mathbf{U} \cdot \exp \mathbf{D} \cdot (\partial_j \mathbf{U}^T) + (\partial_j \mathbf{U}) \cdot \partial_k \exp \mathbf{D} \cdot \mathbf{U}^T + \\ &+ \mathbf{U} \cdot (\partial_{jk} \exp \mathbf{D}) \cdot \mathbf{U}^T + \mathbf{U} \cdot \partial_k \exp \mathbf{D} \cdot (\partial_j \mathbf{U}^T) + \\ &+ (\partial_j \mathbf{U}) \cdot \exp \mathbf{D} \cdot \partial_k \mathbf{U}^T + \mathbf{U} \cdot (\partial_j \exp \mathbf{D}) \cdot \partial_k \mathbf{U}^T + \\ &+ \mathbf{U} \cdot \exp \mathbf{D} \cdot (\partial_{jk} \mathbf{U}^T).\end{aligned}$$

Note that, since the matrix  $\mathbf{D}$  is diagonal, we have

$$\begin{aligned}\partial_j \exp \mathbf{D} &= \exp(\mathbf{D}) \cdot \partial_j \mathbf{D}, \\ \partial_{jk} \exp \mathbf{D} &= \exp(\mathbf{D}) \cdot (\partial_k \mathbf{D} \cdot \partial_j \mathbf{D} + \partial_{jk} \mathbf{D}).\end{aligned}$$

The presented first and second-order derivatives of the metric require the first and second-order spatial derivatives of the eigenvalue decomposition (eigenvalues and eigenvectors), respectively. Their computation is appended in Section 7.

In addition, the derivatives of the eigenvalues and eigenvectors depend on the derivatives of the  $\mathbf{L}$  function presented in Equation (8). In particular, they are given by

$$\nabla \mathbf{L} = \sum_j (\log \mathbf{M}_j) \nabla \hat{N}_j, \quad \nabla^2 \mathbf{L} = \sum_j (\log \mathbf{M}_j) \nabla^2 \hat{N}_j,$$

where  $\nabla$  is the gradient with respect to physical coordinates. Therefore, to differentiate the metric interpolation  $\mathbf{M}(\hat{\mathbf{N}}(\psi(\mathbf{p})))$  at a physical point  $\mathbf{p}$ , the derivatives of the map  $\psi$  presented in Equation (7) and of the shape functions  $\hat{\mathbf{N}}$  are required.

The derivatives of  $\psi|_{E^P}$  are given, at each patch  $E^P \cap E^{\hat{P}}$ , by the ones of the inverse of the physical map  $\phi_{\hat{P}}^{-1}$  corresponding to the background mesh. To obtain the derivatives of the shape functions  $\hat{\mathbf{N}}$  in terms of the physical coordinates  $\mathbf{p}$ , we consider the chain rule for the composition  $\hat{\mathbf{N}} \circ \psi|_{E^P}$  and the restriction of the map  $\psi|_{E^P}$  at each patch  $E^P \cap E^{\hat{P}}$ . We finally obtain the gradient

$$\nabla \hat{\mathbf{N}} = \nabla_{\hat{\xi}} \hat{\mathbf{N}} \cdot \nabla \phi_{\hat{P}}^{-1}, \quad (10)$$

where  $\nabla_{\hat{\xi}}$  is the gradient with respect to  $\hat{\xi}$  coordinates, and the Hessian

$$\nabla^2 \hat{N}_j = \left( \nabla \phi_{\hat{P}}^{-1} \right)^T \cdot \nabla_{\hat{\xi}}^2 \hat{N}_j \cdot \nabla \phi_{\hat{P}}^{-1} + \nabla_{\hat{\xi}} \hat{N}_j \cdot \nabla^2 \phi_{\hat{P}}^{-1}, \quad (11)$$

where

$$\begin{aligned}\nabla \phi_{\hat{P}}^{-1} &= \left( \nabla_{\hat{\xi}} \phi_{\hat{P}} \right)^{-1}, \\ \nabla^2 \phi_{\hat{P}}^{-1} &= \nabla \left( \left( \nabla_{\hat{\xi}} \phi_{\hat{P}} \right)^{-1} \right) = -\nabla \phi_{\hat{P}}^{-1} \cdot \nabla_{\hat{\xi}}^2 \phi_{\hat{P}} \cdot \nabla \phi_{\hat{P}}^{-1}.\end{aligned}$$

## 4. RESULTS

In this section, we present a 2D and a 3D example to illustrate the applicability of our distortion minimization framework for curved  $r$ -adaption to a high-order metric interpolation. First, we generate a background mesh  $\hat{\mathcal{M}}$  and we evaluate the analytical metric  $\mathbf{M}$  at the background mesh nodes. Second, we generate an initial physical mesh  $\mathcal{M}$  and we measure its distortion (quality) by interpolating the metric. Finally, by relocating the nodes, we minimize the mesh distortion problem presented in Equation (6) using the framework presented herein.

To summarize the results, we present a table of the quality statistics, and the figures for the initial and optimized meshes, respectively. Specifically, we show the minimum quality, the maximum quality, the mean quality and the standard deviation of the initial and optimized meshes. We highlight that in all cases, the optimized mesh increases the minimum element quality and it does not include any inverted element. In addition, the meshes resulting after the optimization are composed of elements aligned and stretched to match the target metric tensor. In all figures, the meshes are colored according to the point-wise quality presented in Equation (3).

As a proof of concept, a mesh optimizer has been developed in Julia 1.4.2 [37] with the additional packages: Arpack v0.5.0, Einsum v0.4.1, EllipsisNotation v1.0.0, ILUZero v0.1.0, JLD v0.12.1, Plots v1.9.0, Setfield v0.7.0, SpecialFunctions v1.2.1, StatsBase v0.33.2, TensorOperations v3.1.0 and WriteVTK v1.8.0. In addition, we have used the MATLAB PDE Toolbox [38] to generate the initial isotropic linear unstructured 2D and 3D meshes (the structured meshes are generated by subdivision).

The Julia prototyping code is multithreaded, it corresponds to the implementation of the method presented in this work and the one presented in [21, 24, 25]. In all the examples, the optimization corresponds to finding a minimum of a nonlinear unconstrained multi-variable function. In particular, the mesh optimizer uses an unconstrained line-search globalization with an iterative preconditioned conjugate gradients linear solver. The stopping condition is set to reach an absolute root mean square residual, defined as  $\frac{\|\nabla f(x)\|_{\ell_2}}{\sqrt{n}}$  for  $x \in \mathbb{R}^n$ , smaller than  $10^{-4}$  or a length-step smaller than  $10^{-4}$ . Each optimization process has been performed in a single node of a computing machine. Each node contains two Intel Xeon Platinum 8160 CPU with 24 cores, each at 2.10 GHz, and 96 GB of RAM memory.

We regularize the objective function to ensure infinite values for inverted configurations. Furthermore, to globalize the minimization, we equip Newton's method with a backtracking line-search. Whenever the Newton's update provides an inverted configuration, the objective function becomes infinity and thus, the backtracking line-search shortens the update until a valid configuration is reached.

Following, we first present the target domains to be meshed, and the considered metrics on the domain, Section 4.1. Next, in Section 4.2 we present the optimization results comparing both the proposed discrete based-interpolation procedure and the analytical one from [21, 24, 25]. Finally, in Sections 4.3 and 4.4, we show the application of the discrete metric approach to optimize an anisotropic mesh adapted to a given metric generated by the MMG algorithm presented in [39].

#### 4.1 Domains and metrics

We consider the quadrilateral domain  $\Omega = [-\frac{1}{2}, \frac{1}{2}]^2$  for the two-dimensional examples and the hexahedral domain  $\Omega = [-\frac{1}{2}, \frac{1}{2}]^3$  for the three-dimensional ones. Each domain is equipped with a metric matching a boundary layer. In particular, our target metric  $\mathbf{M}$  is characterized by a diagonal boundary layer metric  $\mathbf{D}$  and a deformation map  $\varphi$  by the following expression

$$\mathbf{M} = \nabla \varphi^T \cdot \mathbf{D} \cdot \nabla \varphi. \quad (12)$$

In what follows, we first detail the boundary layer metric  $\mathbf{D}$  and then the deformation map  $\varphi$ .

The boundary layer aligns with the  $x$ -axis ( $xy$ -plane) in the 2D case (3D case). It determines a constant unit element size along the  $x$ -direction ( $xy$ -directions), and a non-constant element size along the  $y$ -direction ( $z$ -direction). This vertical element size grows linearly with the distance to the  $x$ -axis ( $xy$ -plane), with a factor  $\gamma = 2$ , and starts with the minimal value  $h_{\min} = 0.1$ . Thus, the stretching ratio blends from 1 : 10 to 1 : 1 between  $y = -0.5$  and  $y = 0.5$  (between  $z = -0.5$  and  $z = 0.5$ ). We define the metric for the 2D case as:

$$\mathbf{D} := \begin{pmatrix} 1 & 0 \\ 0 & 1/h(y)^2 \end{pmatrix} \quad (13)$$

where the function  $h$  is defined by

$$h(x) := h_{\min} + \gamma|x|.$$

Similarly, the metric for the 3D case is

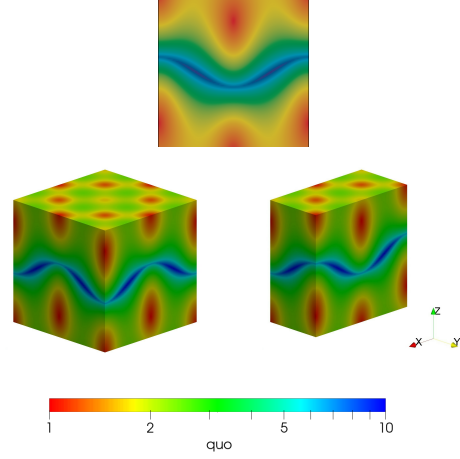
$$\mathbf{D} := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/h(z)^2 \end{pmatrix}. \quad (14)$$

The deformation map  $\varphi$  in Eq. (12) aligns the stretching of  $\mathbf{D}$  according to a given curve in the 2D examples and at a given surface in the 3D examples. In the 2D case, we define the map  $\varphi$  by

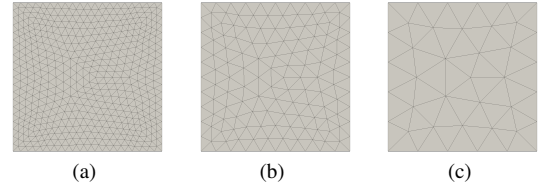
$$\varphi(x, y) = \left( x, \frac{10y - \cos(2\pi x)}{\sqrt{100 + 4\pi^2}} \right),$$

and, in the 3D case by

$$\varphi(x, y, z) = \left( x, y, \frac{10z - \cos(2\pi x) \cos(2\pi y)}{\sqrt{100 + 8\pi^2}} \right).$$



**Figure 4:** Anisotropic quotient values in logarithmic scale of the target metrics.



**Figure 5:** Background triangular meshes of polynomial degree 1, 2 and 4.

Figure 4 shows the anisotropic quotient [40] of the metric presented in Equations (13) and (14). Specifically, the anisotropic quotient of a metric tensor  $\mathbf{M} \in \mathbb{R}^{d \times d}$  is given by

$$\text{quo} = \max_{i=1, \dots, d} \sqrt{\frac{\det(\mathbf{M})}{\lambda_i^d}}$$

where  $\lambda_i$ ,  $i = 1, \dots, d$  are the eigenvalues of  $\mathbf{M}$ . The considered metric  $\mathbf{M}$  attains the highest level of anisotropy, close to the curve described by the points  $(x, y) \in \Omega$  such that  $\varphi(x, y) = (x, 0)$  in 2D, and close the surface described by the points  $(x, y, z) \in \Omega$  such that  $\varphi(x, y, z) = (x, y, 0)$  in 3D.

#### 4.2 Distortion minimization: initial isotropic straight-edged meshes

In this section, we present the optimization results for initially isotropic meshes on the domain equipped with the metrics presented in Section 4.1. We describe first the initial meshes  $\mathcal{M}$  together with the background meshes  $\mathcal{M}_b$  where the metric is interpolated. Next, we present the optimized meshes  $\mathcal{M}^*$  and to conclude, we present the results obtained from the optimization process. Herein, both the background and physical meshes are meshes of the same polynomial degree.

The initial meshes  $\mathcal{M}$  are of polynomial degree 1, 2 and 4.



**Table 1:** Quality Statistics for the initial and optimized meshes with interpolated 2D metric.

Mesh degree	Minimum		Maximum		Mean		Standard deviation	
	Initial	Final	Initial	Final	Initial	Final	Initial	Final
1	0.2066	0.4481	0.9973	0.9853	0.6435	0.7558	0.2149	0.0890
2	0.2608	0.5609	0.9890	0.8647	0.6352	0.7706	0.2087	0.0708
4	0.3504	0.6834	0.9156	0.8268	0.6095	0.7661	0.1877	0.0450

**Table 2:** Quality Statistics for the initial and optimized meshes with analytical 2D metric.

Mesh degree	Minimum		Maximum		Mean		Standard deviation	
	Initial	Final	Initial	Final	Initial	Final	Initial	Final
1	0.2058	0.4510	0.9972	0.9846	0.6443	0.7556	0.2145	0.0823
2	0.2590	0.5648	0.9890	0.8734	0.6351	0.7703	0.2089	0.0715
4	0.3485	0.6838	0.9155	0.8417	0.6096	0.7735	0.1873	0.0530

**Table 3:** Quality Statistics for the initial and optimized meshes with interpolated 3D metric.

Mesh degree	Minimum		Maximum		Mean		Standard deviation	
	Initial	Final	Initial	Final	Initial	Final	Initial	Final
1	0.0875	0.2467	0.9841	0.9594	0.5636	0.6240	0.2199	0.1203
2	0.0980	0.4524	0.9810	0.9118	0.5739	0.6763	0.2214	0.0944
4	0.1929	0.5139	0.9228	0.8289	0.5847	0.7002	0.1998	0.0691

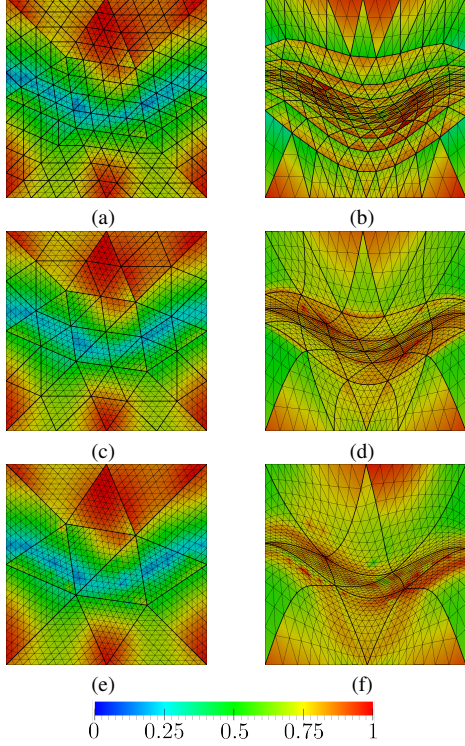
The three meshes feature the same number of nodes and they have the same resolution over the domain. In particular, in 2D the three initial meshes are composed of 481 nodes and 224, 56, and 14 elements, respectively. In 3D, they are composed of 1577 nodes and 7296, 912, and 114 elements, respectively. In Figures 6 and 8 we show the initial meshes, they are colored according to the point-wise stretching and alignment quality measure, presented in Equation (3). Points in blue color have low quality and points with red color have high quality. As we observe, the elements lying in the region of highest stretching ratio have less quality than the elements lying in the isotropic region.

We equip each mesh with the metric presented in Equation (12). We obtain the metric values from the log-Euclidean interpolation method presented in Section 3. In particular, we interpolate the metrics from a background mesh  $\hat{\mathcal{M}}$ , see Figure 5 for the 2D cases. The background meshes are of polynomial degree 1, 2 and 4 according to the polynomial degree of the initial meshes  $\mathcal{M}$ . The three background meshes feature the same number of nodes and they have the same resolution over the domain. In particular, in 2D the three background meshes are composed of 521 nodes and 960, 240 and 60 elements, respectively. In 3D they are composed of 11411 nodes and 59456, 7432 and 929 elements, respectively.

To obtain an optimal configuration  $\mathcal{M}^*$  we minimize the mesh distortion by relocating the mesh nodes while preserving their connectivity, as detailed in Section 2. The coordinates of the inner nodes, and the coordinates tangent to the boundary, are the design variables. Thus, the inner nodes are free to move, the vertex nodes are fixed, while the rest of boundary nodes are enforced to slide along the boundary facets of the domain  $\Omega$ . The total amount of degrees of freedom for the 2D and 3D meshes is 222 and 3957, respectively. In Figure 6 we illustrate the optimized 2D meshes. In the 3D case, Figure 8 shows the interior and exterior of the meshes. We align the axes according to the ones of Figure 4. We observe that the elements lying in the anisotropic region are compressed to attain the stretching and alignment prescribed by the metric.

Tables 1 and 3 show the quality statistics of both the initial and optimized meshes for the 2D and 3D cases, respectively. In all the optimized meshes the minimum is improved and the standard deviation of the element qualities is reduced when compared with the initial configuration.

To validate the proposed method, we compare 2D curved  $r$ -adaption results for the high-order metric interpolation with the results corresponding to an analytic metric evaluation. Considering the initial meshes presented in this section, we optimize the distortion measure by evaluating the analyti-

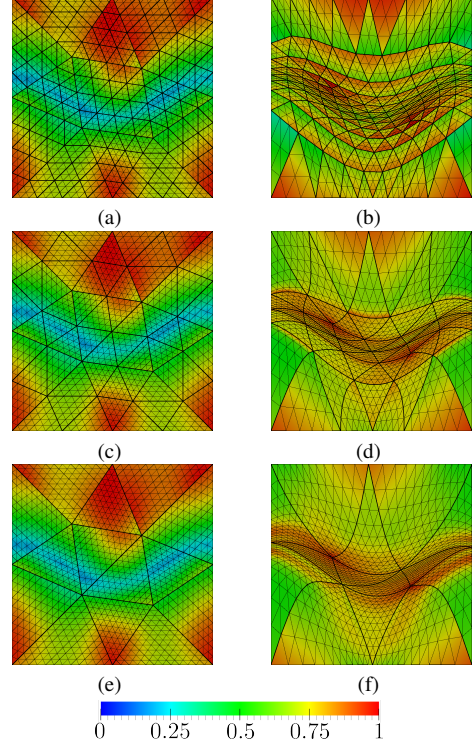


**Figure 6:** Point-wise distortion for triangular meshes of polynomial degree 1, 2 and 4 in rows. Initial straight-sided isotropic meshes and optimized meshes from initial meshes in columns. The interpolation of the metric has been used for the distortion minimization. The sub-triangular elements are the visualization elements. These element vertices are not the high-order degrees of freedom.

cal metric expression, instead of interpolating it in the background mesh. In Figure 7 we show the initial and optimized meshes. They are colored according to the point-wise quality measure of Equation (3) using the analytical metric expression.

To compare quantitatively both results, we compute the relative distance of the node coordinates of the optimized configurations. The relative distance is around  $10^{-2}$  for all the tested cases, obtaining comparable nodal configurations, as it can be observed when comparing Figures 6 and 7.

In Table 2 we present the quality statistics of the initial and optimized meshes using the analytical metric evaluation. To compare the quality improvement of both approaches, we compute the difference between the analyzed quality statistics, obtaining a value for all the statistics below  $10^{-2}$ . Thus, the quality improvement driven by the optimization using the proposed metric interpolation procedure is analogous to the one given by the analytical metric, obtaining in all cases high-quality configurations with a minimum quality over 0.4.

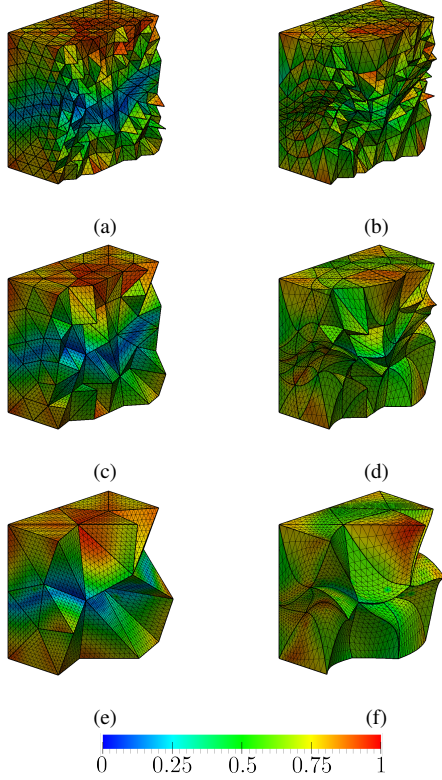


**Figure 7:** Point-wise distortion for triangular meshes of polynomial degree 1, 2 and 4 in rows. Initial straight-sided isotropic meshes and optimized meshes from initial meshes in columns. The analytic evaluation of the metric has been used for the distortion minimization.

### 4.3 Distortion minimization: initial anisotropic straight-edged meshes

The results presented in Section 4.2 show the application of the metric interpolation procedure to optimize isotropic meshes in a domain equipped with a metric. However, in practice, anisotropic meshes are generated combining topological mesh operations that modify the mesh connectivity and mesh r-adaption procedures [6]. To illustrate a practical example, we consider an initial straight-sided mesh adapted by the MMG algorithm presented in [39]. Then, we apply the anisotropic r-adaption method presented in this work.

First, we consider the target metric presented in Equation (12) with  $h_{\min} = 0.01$ . Second, we generate a linear isotropic triangular mesh of input size  $h_{\min}/2 = 0.005$  with MATLAB. Then, we couple such mesh with the target metric evaluated at the mesh vertices and normalized according to different sizes. These sizes are chosen in order to obtain a comparable mesh resolution according to the mesh polynomial degree. Specifically, they are given by 0.0625, 0.125 and 0.25 for the linear, quadratic and quartic case, respectively. We apply the MMG algorithm to obtain a straight-sided anisotropic mesh of polynomial degree 1, 2 and 4, see Figure 9. In particular, they are composed by 1161 nodes and 2137 triangles,



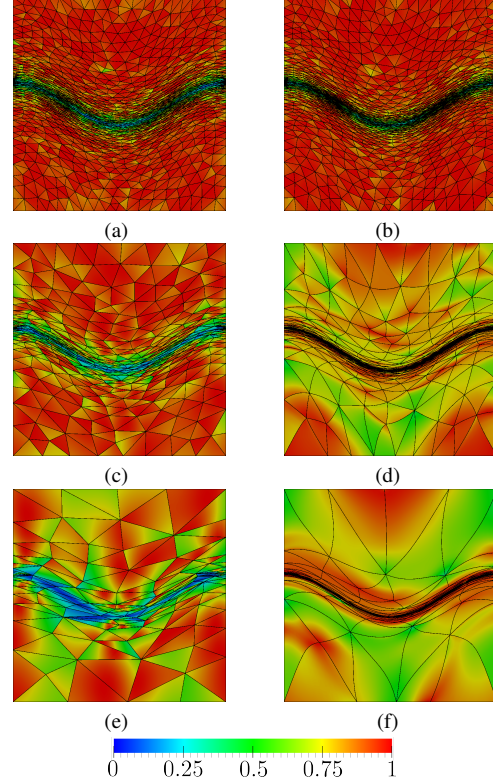
**Figure 8:** Tetrahedral meshes of polynomial degree 1, 2 and 4 in rows. Initial straight-sided isotropic meshes and optimized meshes from initial meshes in columns.

1333 nodes and 624 triangles and, 1525 nodes and 180 triangles, respectively.

The generated meshes are then optimized using the metric interpolation approach presented in this work. In Figure 9 we illustrate the optimized meshes. We observe that the elements lying in the anisotropic region are compressed to attain the stretching and alignment prescribed by the metric. In Table 4 we show the quality statistics of both the initial and optimized meshes. In all the optimized meshes the minimum is improved and the standard deviation of the element qualities is reduced when compared with the initial configuration. We conclude that, with the same metric data and hence, the same inputs, the r-adaption mesh post-processing improves the quality of the meshes generated with the MMG algorithm. In addition, for the straight-edged case, we have presented a global method to improve the stretching and alignment prescribed by the metric after applying an  $h$ -adaption approach.

#### 4.4 Distortion minimization: curved boundaries

We following illustrate that our approach is compatible with curved boundaries. To this end, we consider the holed do-



**Figure 9:** Point-wise distortion for triangular meshes of polynomial degree 1, 2 and 4 in rows. Initial straight-sided anisotropic meshes and optimized meshes from initial meshes in columns.

main  $\Omega = \frac{1}{2}[-1, 1]^2 \setminus C$  where  $C$  is the circle with radius equal to  $\frac{3}{16}$  and centered at the origin. The domain  $\Omega$  has two boundaries, the one of the square  $\frac{1}{2}[-1, 1]^2$  and the one of the circle  $C$ . We equip it with the target metric presented in Equation (12) with  $h_{\min} = 0.01$ . Then, we generate with MMG a linear isotropic triangular mesh of input size  $h_{\min}/2 = 0.005$  over  $\frac{1}{2}[-1, 1]^2$ . As before, we couple such mesh with the target metric evaluated at the mesh vertices and normalized according to size  $h = 0.2$ . Finally, we apply the MMG algorithm to obtain a straight-sided anisotropic mesh of polynomial degree 2 composed by 672 nodes and 290 triangles, see Figures 10 and 11.

To accommodate the curved boundaries we include, to the presented functional, a boundary term that takes into account the mesh approximation to the boundaries of the domain (both the square and the circle). In addition, to approximate the metric stretching, we optimize the mesh using the metric interpolation approach presented in this work. Finally, when optimizing the mesh functional all mesh nodes coordinates are free that is, each mesh node moves in  $\mathbb{R}^2$ .

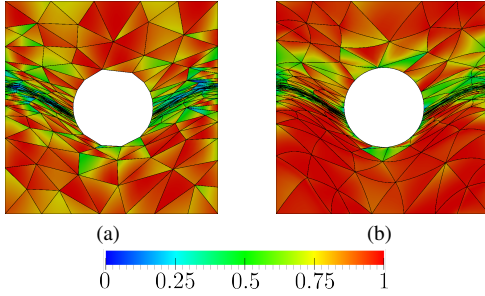
In Figures 10 and 11 we illustrate the optimized mesh. We observe that the elements lying in the anisotropic region are

**Table 4:** Quality Statistics for the initial MMG and optimized meshes with interpolated 2D metric.

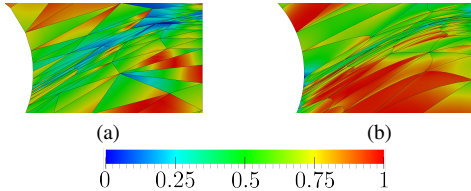
Mesh degree	Minimum		Maximum		Mean		Standard deviation	
	Initial	Final	Initial	Final	Initial	Final	Initial	Final
1	0.0365	0.1794	0.9988	0.9989	0.7806	0.7961	0.2273	0.2040
2	0.0624	0.6300	0.9982	0.9913	0.6966	0.8692	0.2558	0.0788
4	0.0424	0.6063	0.9774	0.9965	0.5677	0.9137	0.2681	0.0886

**Table 5:** Quality Statistics for the initial MMG and optimized mesh with interpolated 2D metric over the holed domain.

Mesh	Minimum	Maximum	Mean	Standard deviation
Initial	0.0489	0.9877	0.6058	0.2512
Optimized	0.3042	0.9927	0.7397	0.1821



**Figure 10:** Point-wise distortion for triangular meshes of polynomial degree 2. Initial straight-sided anisotropic mesh (a) and optimized mesh (b).



**Figure 11:** Zoom of the right region for the initial (a) and optimized mesh (b).

compressed to attain the stretching and alignment prescribed by the metric. Note that the boundary elements are curved to match both the metric and the curved domain boundaries. In Table 5 we show the quality statistics of both the initial and optimized mesh. In the optimized mesh the minimum is improved and the standard deviation of the element qualities is reduced when compared with the initial configuration.

## 5. CONCLUDING REMARKS

In conclusion, we have obtained unique results in curved  $r$ -adaption to a discrete high-order metric. We have rep-

resented the discrete metric in a curved background mesh as a high-order log-Euclidean metric interpolation. For this metric interpolation, we have detailed the first and second derivatives in terms of the physical coordinates. These derivatives have allowed minimizing with Newton's method a mesh distortion accounting for the discrete high-order metric. The discrete metric results compare well with the analytic metric results. In both cases, the method exploits the non-constant Jacobian of curved high-order elements to match curved anisotropic features properly.

In perspective, this capability to match curved anisotropic features might be an attractive ingredient for curved high-order goal-oriented or indicator-based adaption. In these adaptive processes, one would have a high-order metric field in the current curved mesh. This background field would drive curved  $r$ -adaption to globally (locally) relocate the current curved mesh (re-meshed cavity) according to the curved anisotropic features of the solution.

## 6. 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.

## 7. APPENDIX: DERIVATIVES OF THE EIGENVALUE DECOMPOSITION

In this Appendix, we detail the first and second-order spatial derivatives of the eigenvalue decomposition (eigenvalues and eigenvectors), first presented in [36] and rewritten herein using our notation.



Let us consider, for  $\ell = 1, \dots, d$ , the eigenvalue equation for the eigenvector  $\mathbf{u}_\ell$  with eigenvalue  $\lambda_\ell$

$$\mathbf{L}_\ell \mathbf{u}_\ell := (\mathbf{L} - \lambda_\ell \mathbf{I}) \mathbf{u}_\ell = 0,$$

where  $\mathbf{L}$  is a symmetric matrix and  $\mathbf{I}$  is the identity matrix. Then, by taking its first-order and second-order derivatives we respectively obtain

$$0 = \partial_j (\mathbf{L}_\ell \mathbf{u}_\ell) = (\partial_j \mathbf{L}_\ell) \cdot \mathbf{u}_\ell + \mathbf{L}_\ell \cdot \partial_j \mathbf{u}_\ell, \quad (15)$$

$$0 = \partial_{jk} (\mathbf{L}_\ell \mathbf{u}_\ell) = (\partial_{jk} \mathbf{L}_\ell) \cdot \mathbf{u}_\ell + \mathbf{L}_\ell \cdot \partial_{jk} \mathbf{u}_\ell + (\partial_j \mathbf{L}_\ell) \cdot \partial_k \mathbf{u}_\ell + (\partial_k \mathbf{L}_\ell) \cdot \partial_j \mathbf{u}_\ell. \quad (16)$$

For each  $\ell$  one first computes the first-order derivative of the eigenvalue  $\lambda_\ell$  by left-multiplying by  $\mathbf{u}_\ell$  to Equation (15). Then, by solving the remaining unknown term of Equation (15) one obtains the first-order derivatives of the eigenvector  $\mathbf{u}_\ell$ . In particular, the first-order derivatives of the eigenvalues and the eigenvectors are given by

$$\partial_j \lambda_\ell = \mathbf{u}_\ell^T \cdot \partial_j \mathbf{L}_\ell \cdot \mathbf{u}_\ell, \quad \partial_j \mathbf{u}_\ell = -\mathbf{L}_\ell^+ \cdot \partial_j \mathbf{L}_\ell \cdot \mathbf{u}_\ell,$$

where the operation  $\mathbf{L}_\ell^+$  is the Moore-Penrose pseudo-inverse matrix for the matrix  $\mathbf{L}_\ell$ . We use the Moore-Penrose pseudo-inverse matrix instead of the inverse matrix because the matrix  $\mathbf{L}_\ell$  is singular. In addition, the redundant equations are satisfied automatically.

The second-order derivatives are obtained by applying a similar procedure. For each  $\ell$  one first computes the second-order derivative of the eigenvalue  $\lambda_\ell$  by left-multiplying by  $\mathbf{u}_\ell$  to Equation (16). Then, by solving the remaining unknown term of Equation (16) one obtains the second-order derivatives of the eigenvector  $\mathbf{u}_\ell$ . In particular, the second-order derivatives of the eigenvalues are given by

$$\partial_{jk} \lambda_\ell = \mathbf{u}_\ell^T \cdot (\partial_k \mathbf{L}_\ell \cdot \partial_j \mathbf{u}_\ell + \partial_j \mathbf{L}_\ell \cdot \partial_k \mathbf{u}_\ell + \partial_{jk} \mathbf{L}_\ell \cdot \mathbf{u}_\ell),$$

$$\partial_{jk} \mathbf{u}_\ell = -\mathbf{L}_\ell^+ \cdot (\partial_k \mathbf{L}_\ell \cdot \partial_j \mathbf{u}_\ell + \partial_j \mathbf{L}_\ell \cdot \partial_k \mathbf{u}_\ell + \partial_{jk} \mathbf{L}_\ell \cdot \mathbf{u}_\ell) - (\partial_j \mathbf{u}_\ell \cdot \partial_k \mathbf{u}_\ell) \mathbf{u}_\ell,$$

where the last term of the second-order derivative of the eigenvector is obtained by imposing the second-order derivative of the imposed normalization condition  $\mathbf{u}_\ell^T \cdot \mathbf{u}_\ell = 1$

$$0 = \partial_{jk} (\mathbf{u}_\ell^T \cdot \mathbf{u}_\ell) = 2\partial_{jk} \mathbf{u}_\ell^T \cdot \mathbf{u}_\ell + 2\partial_j \mathbf{u}_\ell^T \cdot \partial_k \mathbf{u}_\ell$$

Note that, for each differentiation order, the computation of the eigenvectors derivatives requires the values of the eigenvalues derivatives.

## References

- [1] Yano M., Darmofal D.L. “An optimization-based framework for anisotropic simplex mesh adaptation.” *Journal of Computational Physics*, vol. 231, no. 22, 7626–7649, 2012
- [2] Loseille A., Alauzet F. “Continuous mesh framework part I: well-posed continuous interpolation error.” *SIAM Journal on Numerical Analysis*, vol. 49, no. 1, 38–60, 2011
- [3] Coupez T., Silva L., Hachem E. “Implicit boundary and adaptive anisotropic meshing.” *New Challenges in Grid Generation and Adaptivity for Scientific Computing*, pp. 1–18. Springer, 2015
- [4] Huang W., Russell R.D. *Adaptive Moving Mesh Methods*, vol. 174 of *Applied Mathematical Sciences*. Springer, 2011
- [5] Knupp P.M. “Algebraic mesh quality metrics.” *SIAM J. Numer. Anal.*, vol. 23, no. 1, 193–218, 2001
- [6] Alauzet F., Loseille A. “A Decade of Progress on Anisotropic Mesh Adaptation for Computational Fluid Dynamics.” *Computer-Aided Design, Elsevier*, vol. 72, no. 1, 13–39, 2016
- [7] Gruau C., Coupez T. “3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric.” *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 48-49, 4951–4976, 2005
- [8] Frey P., Alauzet F. “Anisotropic mesh adaptation for CFD computations.” *Computer methods in applied mechanics and engineering*, vol. 194, no. 48-49, 5068–5082, 2005
- [9] Hecht F. “BAMG: bidimensional anisotropic mesh generator.” *User Guide. INRIA, Rocquencourt*, 1998
- [10] Coupez T. “Metric construction by length distribution tensor and edge based error for anisotropic adaptive meshing.” *Journal of computational physics*, vol. 230, no. 7, 2391–2405, 2011
- [11] Fidkowski K.J., Darmofal D.L. “Review of output-based error estimation and mesh adaptation in computational fluid dynamics.” *AIAA journal*, vol. 49, no. 4, 673–694, 2011
- [12] Coupez T. “On a Basis Framework for High Order Anisotropic Mesh Adaptation.” vol. 203, 141–153, 2017. Research Note 26th International Meshing Roundtable
- [13] Johnen A., Geuzaine C., Toulorge T., Remacle J.F. “Quality Measures for Curvilinear Finite Elements.” *TILDA: Towards Industrial LES/DNS in Aeronautics*, p. 221, 2021
- [14] Zhang R., Johnen A., Remacle J.F. “Curvilinear mesh adaptation.” *International Meshing Roundtable*, pp. 57–69. Springer, 2018

- [15] Zahr M.J., Persson P.O. “An optimization based discontinuous Galerkin approach for high-order accurate shock tracking.” *2018 AIAA Aerospace Sciences Meeting*, p. 0063. 2018
- [16] Zahr M.J., Shi A., Persson P.O. “Implicit shock tracking using an optimization-based high-order discontinuous Galerkin method.” *Journal of Computational Physics*, vol. 410, 109385, 2020
- [17] Zahr M.J., Persson P.O. “An r-adaptive, high-order discontinuous Galerkin method for flows with attached shocks.” *AIAA Scitech 2020 Forum*, p. 0537. 2020
- [18] Marcon J., Turner M., Moxey D., Sherwin S.J., Peiró J. “A variational approach to high-order r-adaptation.” *IMR26*, 2017
- [19] Marcon J., Castiglioni G., Moxey D., Sherwin S.J., Peiró J. “rp-adaptation for compressible flows.” *International Journal for Numerical Methods in Engineering*, vol. 121, no. 23, 5405–5425, 2020
- [20] Dobrev V., Knupp P., Kolev T., Mittal K., Tomov V. “The target-matrix optimization paradigm for high-order meshes.” *SIAM Journal on Scientific Computing*, vol. 41, no. 1, B50–B68, 2019
- [21] Aparicio-Estrems G., Gargallo-Peiró A., Roca X. “Defining a Stretching and Alignment Aware Quality Measure for Linear and Curved 2D Meshes.” *International Meshing Roundtable*, pp. 37–55. Springer, 2018
- [22] Dobrev V., Knupp P., Kolev T., Tomov V. “Towards simulation-driven optimization of high-order meshes by the Target-Matrix Optimization Paradigm.” *International Meshing Roundtable*, pp. 285–302. Springer, 2018
- [23] Dobrev V., Knupp P., Kolev T., Mittal K., Rieben R., Tomov V. “Simulation-driven optimization of high-order meshes in ALE hydrodynamics.” *Computers & Fluids*, vol. 208, 104602, 2020
- [24] Aparicio-Estrems G., Gargallo-Peiró A., Roca X. “Anisotropic Optimization of curved meshes: specific-purpose line-search and trust-region globalizations for Newton’s method.” *International Meshing Roundtable*. 2019
- [25] Aparicio-Estrems G., Gargallo-Peiró A., Roca X. “Stretching and aligning piece-wise polynomial meshes to match curved anisotropic features.” *International Conference on Spectral and High-Order Methods*. 2021
- [26] Arsigny V., Fillard P., Pennec X., Ayache N. “Log-Euclidean Metrics for Fast and Simple Calculus on Diffusion Tensors.” *Magnetic Resonance in Medicine*, vol. 56, 411–421, 2006
- [27] Rochery L., Loseille A. “P2 cavity operator and Riemannian curved edge length optimization: a path to high-order mesh adaptation.” *AIAA Scitech 2021 Forum*, p. 1781. 2021
- [28] Branets L.V., Garanzha V.A. “Distortion measure of trilinear mapping. Application to 3-D grid generation.” *Numerical linear algebra with applications*, vol. 9, no. 6-7, 511–526, 2002
- [29] López E.J., Nigro N.M., Storti M.A. “Simultaneous untangling and smoothing of moving grids.” *Int. J. Numer. Meth. Eng.*, vol. 76, no. 7, 994–1019, 2008
- [30] Escobar J.M., Rodríguez E., Montenegro R., Montero G., González-Yuste J.M. “Simultaneous untangling and smoothing of tetrahedral meshes.” *Comput. Meth. Appl. Mech. Eng.*, vol. 192, no. 25, 2775–2787, 2003
- [31] Gargallo-Peiró A., Roca X., Peraire J., Sarrate J. “Optimization of a regularized distortion measure to generate curved high-order unstructured tetrahedral meshes.” *Int. J. Numer. Meth. Eng.*, vol. 103, 342–363, 2015
- [32] Gargallo-Peiró A. *Validation and generation of curved meshes for high-order unstructured methods*. Ph.D. thesis, Universitat Politècnica de Catalunya, 2014
- [33] Ekelschot D., Ceze M., Murman S.M., Garai A. “Parallel high-order anisotropic meshing using discrete metric tensors.” *AIAA Scitech 2019 Forum*, p. 1993. 2019
- [34] Mittal K., Dutta S., Fischer P. “Nonconforming Schwarz-spectral element methods for incompressible flow.” *Computers & Fluids*, vol. 191, 104237, 2019
- [35] Sitaraman J., Floros M., Wissink A., Potsdam M. “Parallel domain connectivity algorithm for unsteady flow computations using overlapping and adaptive grids.” *Journal of Computational Physics*, vol. 229, no. 12, 4703–4723, 2010
- [36] Andrew A.L., Chu K.W.E., Lancaster P. “Derivatives of eigenvalues and eigenvectors of matrix functions.” *SIAM journal on matrix analysis and applications*, vol. 14, no. 4, 903–926, 1993
- [37] Bezanson J., Edelman A., Karpinski S., Shah V.B. “Julia: A fresh approach to numerical computing.” *SIAM review*, vol. 59, no. 1, 65–98, 2017
- [38] MATLAB. *version 9.3.0.713579 (R2017b)*. The MathWorks Inc., Natick, Massachusetts, 2017
- [39] Dobrzynski C. *MMG3D: User guide*. Ph.D. thesis, INRIA, 2012
- [40] Loseille A., Löhner R. “Anisotropic Adaptive Simulations in Aerodynamics.” *AIAA 2010-169*, 2010. 49th AIAA Aerospace Sciences Meeting, Fairfax, Va, USA

# AN EFFICIENT SOLVER TO APPROXIMATE CAD CURVES WITH SUPER-CONVERGENT RATES

Julia Docampo-Sánchez

Eloi Ruiz-Gironés

Xevi Roca

*Barcelona Supercomputing Centre-Centro Nacional de Supercomputacin BSC-CNS, Spain.  
julia.docampo@bsc.es, eloi.ruiz@bsc.es, xevi.roca@bsc.es*

## ABSTRACT

We present a specific-purpose solver to approximate curves with super-convergent rates. To obtain super-convergence, we minimize a disparity measure in terms of a piece-wise polynomial approximation and a curve re-parametrization. We have numerical evidence that the disparity converges with  $2p$  order for planar curves and  $\lfloor \frac{3}{2}(p-1) \rfloor + 2$  for 3D curves,  $p$  being the mesh polynomial degree. To meet these rates, we exploit the quadratic convergence of a globalized Newton’s method with the help of three main ingredients. First, we employ a nonmonotone line search reducing the number of nonlinear iterations. The second ingredient is to introduce a log barrier function preventing element inversion in the curve re-parameterization. Third, we propose a constrained optimization of the disparity functional where the element interfaces are fixed, improving the computational efficiency whilst preserving super-convergence. We approximate analytic curves as well as CAD models with meshes of several polynomial degrees. We conclude that the solver is well-suited to obtain super-convergent approximations to curves at reasonable computational times.

**Keywords:** curve approximation, distance optimization, super-convergence, high-order meshes

## 1. INTRODUCTION AND MOTIVATION

Geometric accuracy plays a major role in the performance of unstructured high-order methods [1], requiring curved elements to meet the desired accuracy. The geometric accuracy is measured as the distance between the mesh and the target geometry. Traditionally, in computational geometry, this corresponded to the Fréchet and Hausdorff distances [2]. More recently, distance optimization techniques have been proposed. For example, in [3, 4] an area and Taylor based distance optimizer are used respectively for 2D and 3D geometry. The authors report significant mesh-CAD distance reductions at adequate computational times.

In addition, a disparity measure for generating optimal curved high-order meshes was proposed in [5]. The optimization combines a distortion measure for mesh quality and a geometric  $L^2$ -disparity measure for geometric error [6]. It produces optimal non-interpolative meshes and it has been observed that this disparity is  $2p$  super-convergent [7]. This affords a straightfor-

ward advantage: one can obtain the desired geometric accuracy using smaller polynomial degrees than with standard interpolation approaches.

As in many optimization problems, the original disparity was solved with a Newton method combined with Armijo backtracking line search. The Armijo rule is monotonic: the iteration is valid if the objective function decreases. For highly nonlinear problems, monotonicity can trap the optimizer if it follows a narrow curved valley (making very short steps or zigzagging) and reduces the convergence rate [8]. This has motivated the development of nonmonotone line searches.

The first nonmonotone line search was based on a maximum principle [9]. A step is considered valid if it improves the objective function with respect to the maximum value of the objective functions corresponding to the  $N$  previous iterations. Later, a family of nonmonotone line searches were proposed where a tuning parameter shifts the search condition from maximum to an average [10]. The authors prove global conver-

gence for this family of line search methods. In fact, nonmonotone line searches have demonstrated an improvement in computational efficiency, as well as likelihood of finding a global minimum [9, 11].

The work presented here focuses on improving the computational performance during the optimization of the disparity measure. Focusing on curves, we propose:

1. A reduction of the optimization dimension by fixing the element interfaces.
2. A logarithmic barrier preventing curve tangling.
3. An average based (nonmonotone) line search.

Our results show that switching to a nonmonotone line search consistently reduces the number of iterations. Furthermore, by fixing the element interfaces we reduce the dimension of the optimization problem. Although this constrained version is sub-optimal in terms of the error compared to the original disparity (free interfaces), we are able to preserve the same super-convergent property. We have performed numerical tests based on analytic curves as well as CAD geometry obtained through ESP [12].

This paper is organized as follows. In Section 2, we define the disparity measure for curves and show the super-convergent property through an example. In Section 3, we discuss the disparity and constrained optimization and the new solver features. Finally, in Sections 4 and 5, we compare the errors and iterations from the constrained and unconstrained problem studying several CAD geometries. The paper concludes with Section 6 discussing main results.

## 2. DISPARITY MEASURE: CURVES

We begin by introducing the disparity formulation for curves. We will discuss how it is defined, optimized and implemented as well as show how it attains  $2p$  order. For more details, we refer the reader to [7].

### 2.1 Mathematical formulation

We define our mesh as a set of elements where for each physical element  $e^P$  there is a reference element  $e^R$ . The physical mesh  $\mathcal{M}^P$  can be defined in terms of an element-wise parametrization  $\phi^P$ :

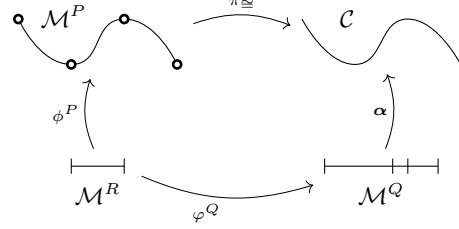
$$\phi^P|_{e^R} : e^R \rightarrow e^P \subset \mathbb{R}^n \quad (1)$$

$$\xi \rightarrow \mathbf{x} = \sum_{i=1}^{p+1} \mathbf{x}_i N_i^p(\xi), \quad (2)$$

where  $p+1$  are the number of nodes for the high-order element  $e^P$ ,  $\mathbf{x}_i$  the  $\mathbb{R}^n$ -physical coordinates of the  $i$ -th node and  $\{N_i^p\}_{i=1}^{p+1}$  a Lagrangian basis of degree  $p$ .

Let  $\mathcal{C} \subset \mathbb{R}^n$  be a curve parametrized by  $\alpha : [a, b] \subset \mathbb{R} \rightarrow \mathcal{C}$  and consider the family of mappings:

$$\Pi := \left\{ \pi \in \mathcal{H}^1(\mathcal{M}^P, \mathcal{C}), \pi \text{ diffeomorphism} \right\}.$$



**Figure 1:** commutative diagram showing the reference mesh  $\mathcal{M}^R$ , the mappings to the physical meshes:  $\phi^P$  and  $\phi^Q$  respectively, the curve  $\mathcal{C}$  and its parametrization  $\alpha$  and the projection  $\pi$ .

The diagram in Figure 1 shows the reference mesh  $\mathcal{M}^R$ , the physical mesh  $\mathcal{M}^P$  consisting of three elements, the target curve  $\mathcal{C}$  and the diffeomorphism  $\pi$ . The disparity measure between the high-order mesh and the curve is the minimal projection error

$$d(\mathcal{M}^P, \mathcal{C}) = \inf_{\pi \in \Pi} \left( \int_{\mathcal{M}^P} |\mathbf{x} - \pi(\mathbf{x})|^2 d\mathbf{x} \right)^{1/2}, \quad (3)$$

where  $|\cdot|$  denotes the Euclidean norm of vectors. Defining the functional

$$E(\mathbf{x}, \pi) = \int_{\mathcal{M}^P} |\mathbf{x} - \pi(\mathbf{x})|^2 d\mathbf{x} \quad (4)$$

$$= \int_{\mathcal{M}^R} |\phi^P(\xi) - \pi \circ \phi^P(\xi)|^2 |\dot{\phi}^P(\xi)| d\xi \quad (5)$$

$$= \|\phi^P - \pi \circ \phi^P\|_{\sigma}^2, \quad (6)$$

we establish the following relation:

$$d(\mathcal{M}^P, \mathcal{C})^2 = \inf_{\pi \in \Pi} E(\mathbf{x}, \pi). \quad (7)$$

Notice that we use the sub-index  $\sigma$  to denote that it is integral with weight  $|\dot{\phi}^P|$ .

Consider any possible curve reparametrization:  $\alpha \circ s$ . As in (1), we define a 1D mesh through the mapping:

$$\varphi^Q|_{e^R} : e^R \rightarrow e^Q \subset \mathbb{R} \quad (8)$$

$$\xi \rightarrow s = \sum_{i=1}^{q+1} s_i \cdot N_i^q(\xi). \quad (9)$$

**Note 1** The mesh  $\phi^P$  is in the physical space whereas  $\varphi^Q$  is a mesh in the parametric space. Modifying  $\varphi^Q$  results in different curve parametrizations.

As shown in Figure 1, we have that  $\pi \circ \phi^P = \alpha \circ \varphi^Q$ . Hence, we can reformulate the problem:

$$E(\mathbf{x}, \pi) = E(\mathbf{x}, s) = \|\mathbf{x} - \alpha \circ s\|_\sigma^2 \quad (10)$$

$$= \int_{\mathcal{M}^R} |\phi^P(\xi) - \alpha \circ \varphi^Q(\xi)|^2 |\phi^P(\xi)| d\xi. \quad (11)$$

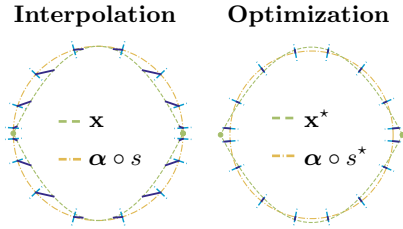
Therefore, the mapping  $s : \mathcal{M}^R \rightarrow \mathcal{M}^Q$  needs to be a diffeomorphism, too. In Section 3, we show how to enforce this numerically.

Optimizing  $E$  with respect to  $s$  gives the disparity between the mesh and the curve. If we optimize  $E$  with respect to both  $\mathbf{x}$  and  $s$ , we obtain the mesh with optimal geometric accuracy according to the disparity measure. We define the optimal approximation as:

$$\mathbf{x}^*, s^* = \arg \min_{\mathbf{x}, s} \|\mathbf{x} - \alpha \circ s\|_\sigma^2 \quad (12)$$

$$= \arg \min_{\mathbf{x}, s} \int_{\mathcal{M}^R} |\mathbf{x}(\xi) - \alpha \circ s(\xi)|^2 |\dot{\mathbf{x}}(\xi)| d\xi. \quad (13)$$

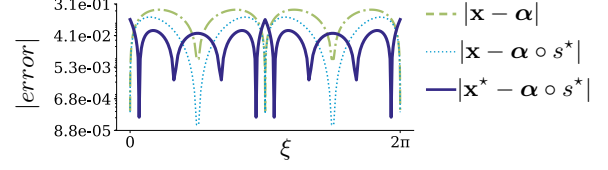
**Remark 1** The optimal  $s^*$  (with respect to  $\mathbf{x}^*$ ) minimizes the error in the tangent direction. In Figure 2, we draw the point-wise errors starting with an interpolative mesh. After optimization, we see that the errors align with the curve normal direction.



**Figure 2:** a circle meshed with two  $p = 2$  elements showing the point-wise errors (solid deep blue lines). The curve normal direction is denoted by dashed light blue lines.

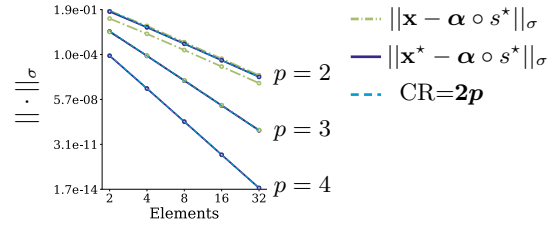
## 2.2 An example of super-convergence

Let us discuss the role played by the physical ( $\mathbf{x}$ ) and parametric ( $s$ ) meshes. Figure 3 shows several point-wise errors: first, we approximate the circle with interpolative meshes:  $(\mathbf{x}_0, s_0)$ . Then, we compute the disparity measure of this mesh optimizing the parametric mesh,  $s^* = \arg \min_s \|\mathbf{x} - \alpha \circ s\|_\sigma^2$ . Finally, we optimize both:  $\mathbf{x}^*, s^* = \arg \min_{\mathbf{x}, s} \|\mathbf{x} - \alpha \circ s\|_\sigma^2$ . Notice that the optimal mesh significantly improves the geometric error.



**Figure 3:** point-wise errors approximating a circle with two elements of degree  $p = 2$  obtained with: direct interpolation  $|\mathbf{x}(\xi) - \alpha(\xi)|$  (dashed green), the disparity  $|\mathbf{x}(\xi) - \alpha \circ s^*(\xi)|$  (dotted light blue) and the optimized disparity  $|\mathbf{x}^*(\xi) - \alpha \circ s^*(\xi)|$  (solid deep blue).

In Figure 4, we show convergence plots of the disparity measure when approximating the same circle using meshes of degree  $p = 2, 3, 4$  and for several  $h$ -refinements. The initial disparity gives to  $p + 1$  order. On the other hand, the slope of the optimal pair  $(\mathbf{x}^*, s^*)$  shows a convergence order of  $2p$ .



**Figure 4:** slopes (log-log) of the disparity approximating a circle for several mesh refinements showing the convergence rates (CR) before  $(\mathbf{x}, s^*)$  and after  $(\mathbf{x}^*, s^*)$  optimizing the disparity measure.

## 2.3 Optimization challenges

The solution  $(\mathbf{x}^*, s^*) = \arg \min_{\mathbf{x}, s} E(\mathbf{x}, s)$  is found with a monotone backtracking line search:

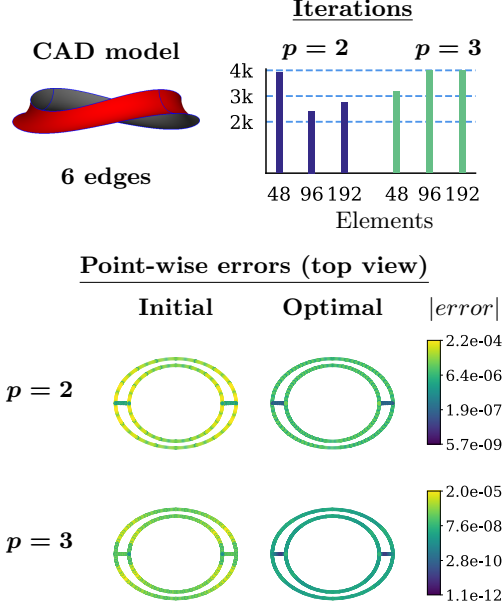
$$(\mathbf{x}_{n+1}, s_{n+1}) = (\mathbf{x}_n, s_n) + \alpha \delta_n, \quad (14)$$

where  $\delta_n$  is a Newton step and  $\alpha \in (0, 1]$  is such that

$$E_{n+1} < E_n + \alpha 10^{-4} \delta_n \cdot \nabla E_n \quad (\text{Armijo rule}). \quad (15)$$

This rule ensures that the objective function (disparity) decreases in every step. However, it has an impact on the number of iterations. Furthermore, decreasing  $E_n$  does not assure valid solutions; since  $s$  is unconstrained,  $\alpha \circ s$  can tangle during minimization.

In Figure 5, we show the optimization results for the six edges of a simple CAD body obtained from ESP [12]. We study the number of iterations (top) as well as the point-wise errors (bottom). The point-wise error is computed as  $|\mathbf{x} - \alpha \circ s|$  (initial) and  $|\mathbf{x}^* - \alpha \circ s^*|$  (optimal), respectively. Observe that although the error improves, the number of iterations is large, reaching almost 4000 in several cases.



**Figure 5:** optimal disparity using the Armijo rule for the 6 edges of a CAD model and meshes consisting of 8 elements (degrees  $p = 2, 3$ ). The plots show the iterations taken by the optimizer (top) and the initial and optimal point-wise errors respectively (bottom).

This paper addresses these computational issues. First, we propose reducing the dimension of the optimization problem by defining a constrained disparity functional with fixed element interfaces. Second, we insert a logarithmic barrier preventing  $s^*$  from tangling. Last, we introduce the average line search in [10] reducing the number of nonlinear iterations.

### 3. NEW SOLVER: CONSTRAINED OPTIMIZATION, LOG BARRIER AND AVERAGE LINE SEARCH

We now present a modified optimization approach for the disparity. We give an analogous mathematical formulation and highlight the differences with respect to the original method. Then, we focus on the solver details and introduce two major modifications: the log barrier function which effectively prevents curves from tangling and the Zhang-Hager [10] line search.

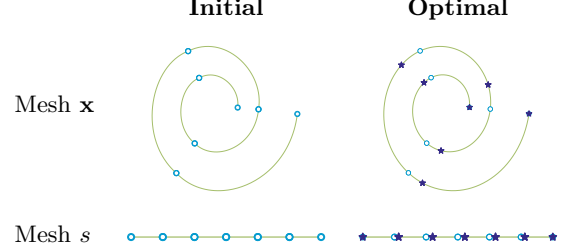
#### 3.1 Optimizing the constrained disparity

To optimize the geometric accuracy of a high-order mesh, we solve the problem:

$$E(\mathbf{x}^*, s^*) = \min_{\mathbf{x}, s} \|\mathbf{x} - \alpha \circ s\|_{\sigma}^2,$$

with  $\mathbf{x}$  consisting of elements of degree  $p$  and  $s$  elements of degree  $q$ . Since we define our elements as

all possible mappings from the reference to the physical element:  $\phi^P : e^R \rightarrow e^P$ , we are considering all possible partitions along the curve. Thus, the optimal mesh according to the disparity alters the initial element partition. This is illustrated in Figure 6 where we have optimized a spiral curve. The initial element configuration both in  $\mathbf{x}$  and  $s$  changes after optimization, moving elements towards the spiral end.



**Figure 6:** a spiral approximated with 6 elements (interfaces denoted by  $\circ, \star$ ) showing the meshes  $\mathbf{x}$  and  $s$  before and after optimizing the disparity.

Assume now a fixed element configuration and, at each element, define the meshes by

$$\tilde{\mathbf{x}}(\xi) = \underbrace{\mathbf{x}_0 N_0^p(\xi) + \mathbf{x}_p N_{p+1}^p(\xi)}_{\mathbf{x}_F(\xi)} + \sum_{i=2}^p \tilde{\mathbf{x}}_i \cdot N_i^p(\xi), \quad (16)$$

$$\tilde{s}(\xi) = \underbrace{s_0 N_0^q(\xi) + s_q N_{q+1}^q(\xi)}_{s_F(\xi)} + \sum_{i=2}^q \tilde{s}_i \cdot N_i^q(\xi), \quad (17)$$

where  $\mathbf{x}_0$ ,  $\mathbf{x}_{p+1}$ ,  $s_0$ ,  $s_{q+1}$  are fixed throughout the optimization. Note that the indices run from 2 to  $p$  and 2 to  $q$  respectively, excluding the first and last nodes. We define the optimal *constrained* mesh as

$$\tilde{\mathbf{x}}^*, \tilde{s}^* = \arg \min_{\tilde{\mathbf{x}}, \tilde{s}} \|\tilde{\mathbf{x}} - \alpha \circ \tilde{s}\|_{\sigma}^2. \quad (18)$$

Note that the overall accuracy is dictated by the initial element distribution. A uniform parametric partition on the CAD may lead to a poor element distribution. When a curvature-based mesher is not available, we propose a pre-processing stage: first, optimize the unconstrained disparity functional and obtain the linear meshes  $(\mathbf{x}_1^*, s_1^*)$  with optimal (in the disparity sense) element distribution. Then, we  $p$ -refine using  $s_1^*$ :

- For each element in the parametric mesh, we create a high-order element in the physical mesh.
- For each element of the parametric mesh, the nodes of the physical mesh are created as

$$\mathbf{x}_i = \alpha \circ s_1^*(\xi_i), \quad i = 1, \dots, p+1,$$

where  $\xi_i$  is a distribution of high-order nodes in the master element.



Finally, we fix the high-order element interfaces and optimize the constrained disparity functional. In Section 4, we show an example of how the initial interpolative meshes benefit from this pre-processing stage.

### 3.2 Logarithmic barrier

The commutative diagram shown in Figure 1 holds if all mappings are diffeomorphisms. Since there are no constraints in formulation (12), in particular, diffeomorphism  $s$  is not actively enforced. The curve will tangle if elements in the parametric mesh  $s$  are inverted. A possible solution is to add a constraint on  $s'$  through the line search [13, 14]. Alternatively, we can avoid curve tangling by introducing a log barrier function.

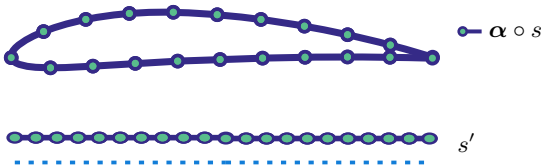
**log barrier [15, Ch.9]:** consider the nonlinear programming problem:

$$\min f(x) \quad \text{subject to} \quad c_i \geq 0, \quad i = 1, \dots, m.$$

The logarithmic barrier is a *penalty* term that moves the optimizer away from violating any of the constraints  $c_i$ . For a given  $\mu$ , one can solve instead

$$\min_x P(x; \mu) = \min_x f(x) - \mu \sum_{i=1}^m \log(c_i(x)).$$

A suitable curve parametrization should follow the curve either forward or backwards. When we compose  $\alpha \circ s$ , we can ensure that the reparametrization preserves direction by fixing the sign of  $s'$  (positive being forward and negative backwards) at the beginning of the optimization. For example, in Figure 7 we show a NACA curve and the initial parametrization  $\alpha \circ s = \alpha$ . Note that since  $s$  has not been optimized, it behaves as a linear mapping with a constant derivative.



**Figure 7:** initial parametrization of a NACA curve and the  $s'$  profile with respect to the zero axis (...).

Regarding the disparity (either constrained or unconstrained), we want to solve:

$$\min_{\mathbf{x}, s} E(\mathbf{x}, s) \quad \text{subject to} \quad s'(\xi) > 0 \quad \forall \xi.$$

**Remark 2** In this case,  $E$  can be used to obtain either the original  $(\mathbf{x}^*, s^*)$  or the constrained  $(\tilde{\mathbf{x}}^*, \tilde{s}^*)$  solutions. Both solutions benefit from this technique.

We need a continuous barrier function so we introduce:

$$P(\mathbf{x}, s; \mu) = E(\mathbf{x}, s) - \mu \int_{\mathcal{M}^R} \log(s'(\xi)) d\xi. \quad (19)$$

For each  $\mu$ , we optimize instead the following:

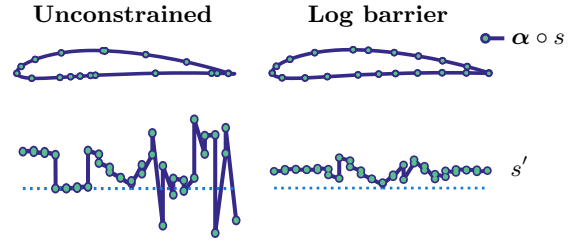
$$(\mathbf{x}^*, s^*) = \arg \min_{\mathbf{x}, s} P(\mathbf{x}, s; \mu), \quad (20)$$

and use that if  $s$  is not tangled, then

$$\lim_{\mu \rightarrow 0} P(\mathbf{x}, s; \mu) = E(\mathbf{x}, s).$$

**Note 2** In practise, the log-barrier is activated only if at a particular Newton step, the solver detects a change in the sign of  $s'$ . This check is done oversampling  $s$  at each element. At that point, it retrieves the previous valid pair  $(\mathbf{x}_n, s_n)$  and solves instead the penalized problem  $P(\mathbf{x}, s; \mu_k)$ ,  $k = 1, \dots, M$ .

In Figure 8 (left), we show the optimized NACA curve  $\alpha \circ s$  without a log barrier where the curve develops artificial loops. Observe how the derivative profile  $s'$  crosses the zero axis in several locations. On the right, we show the same curve but optimized with the log-barrier, resulting in a valid curve.



**Figure 8:** untangling the NACA curve. Optimizing with and without log barrier. (...) denotes  $s' = 0$ .

### 3.3 Zhang-Hager line search

We find the solution to our minimization problem combining Newton with backtracking line search:

$$(\mathbf{x}_{n+1}, s_{n+1}) = (\mathbf{x}_n, s_n) + \alpha_n \mathbf{d}_n. \quad (21)$$

Let  $\nabla(\cdot)$  and  $H(\cdot)$  denote the gradient and Hessian operators respectively and  $\delta_n$  a Newton step:

$$H(E_n) \delta_n = -\nabla(E_n). \quad (22)$$

At each iteration, we ensure a descent direction using:

$$\mathbf{d}_n := \begin{cases} \delta_n, & \text{if } \delta_n \cdot \nabla E_n < 0, \\ -[\text{diag}(H(E_n))]^{-1} \nabla E_n, & \text{otherwise.} \end{cases} \quad (23)$$

As mentioned before, the standard line search choice is the *Armijo* (monotone) rule:  $\alpha \in (0, 1]$  satisfies that

$$E_{n+1} < E_n + \alpha 10^{-4} \mathbf{d}_n \cdot \nabla E_n. \quad (24)$$

Instead, we use a type of Zhang-Hager nonmonotone line search [10]. Let  $C_0 = E_0$ ,  $Q_0 = 1$  and define:

$$Q_{n+1} = \eta_n Q_n + 1 \quad C_{n+1} = \frac{\eta_n Q_n C_n + E_{n+1}}{Q_{n+1}}, \quad (25)$$

Notice that  $\eta \equiv 0$  gives Armijo's monotone line search. On the other hand,  $\eta \equiv 1$  gives an average based rule:

$$C_n = \frac{1}{n} \sum_{i=1}^n E_i, \quad (26)$$

which is the one that we will use in our experiments. Finally,  $\alpha_n$  satisfies **Wolfe conditions**:

$$E_{n+1} \leq C_n + \sigma_1 \alpha_n \nabla E_n \cdot \mathbf{d}_n \quad (27)$$

$$\nabla E_{n+1} \cdot \mathbf{d}_n \geq \sigma_2 \nabla E_n \cdot \mathbf{d}_n \quad (28)$$

with  $\sigma_1 = 10^{-4}$  and  $\sigma_2 = 0.9$  as suggested in [15].

### 3.4 Algorithm

We provide the implementation details of the proposed solver for the optimization of the constrained and unconstrained disparity measure, see Algorithm 1.

For the constrained problem, we solve the nonlinear optimization problem:

$$\tilde{E}(\tilde{\mathbf{x}}^*, \tilde{s}^*) = \min_{\tilde{\mathbf{x}}, \tilde{s}} \|\tilde{\mathbf{x}} - \boldsymbol{\alpha} \circ \tilde{s}\|_\sigma, \quad (29)$$

where  $\tilde{\mathbf{x}}$  and  $\tilde{s}$  have the element interfaces fixed.

The main function is OPTIMIZE. It takes several arguments:  $\boldsymbol{\alpha}$  gives information about the curve. Parameters  $p$  and  $q$  denote the polynomial degrees in  $\mathbf{x}$  and  $s$ , respectively.  $M$  is the number of outer iterations corresponding to the log barrier term  $\mu$  and  $N$  the maximum nonlinear iterations allowed. In our experiments, we use  $M=6$  with  $\mu$  decreasing by a factor of  $10^{-2}$  at each step. The parameter optimizePartitions indicates whether or not the initial element partition should be optimized (using the original disparity with  $p = q = 1$ ). Finally, freeInterfaces is a flag indicating if the element interfaces are fixed or not.

At the start (lines 2-5), we check if the CAD model has already an initial tessellation. Otherwise, we create an element partition. Then, if optimizePartitions is activated, we set  $p = q = 1$  and call again the function optimize with the flag freeInterfaces activated (lines 6-7). In this case, the same routine follows but changes  $\tilde{E}$  (constrained) to  $E$  (original disparity). Next (line

---

#### Algorithm 1 constrained disparity minimization

---

```

1: function OPTIMIZE( $\boldsymbol{\alpha}$ ,  $p$ ,  $q$ ,  $M$ ,  $N$ , optimizeParti-
   tion, freeInterfaces)
2:   if  $\boldsymbol{\alpha}.tess = true$  then
3:      $r = \boldsymbol{\alpha}.tess$ 
4:   else
5:      $r = divide(\boldsymbol{\alpha}, n)$ 
6:   if optimizePartition = true then
7:      $r, \_ \leftarrow OPTIMIZE(\boldsymbol{\alpha}, 1, 1, M, N, false, true)$ 
8:    $\tilde{\mathbf{x}}_0, \tilde{s}_0 \leftarrow REFINE(\boldsymbol{\alpha}, r, p, q)$ 
9:    $\mu = 0$ ; logBarrier = false;
10:  for  $m = 1 : M$  do
11:    if  $m = M$  then
12:       $\mu = 0$ 
13:    else
14:       $\mu = \mu \cdot 10^{-2}$ 
15:    for  $n = 1 : N$  do
16:       $\nabla E_n, H(E_n) \leftarrow GRADHESS(\boldsymbol{\alpha}, \mathbf{x}_n, s_n,$ 
        freeInterfaces,  $\mu_k)$ 
17:      if  $|\nabla(E_n)| < tol$  then
18:        break
19:       $\boldsymbol{\delta} = -H(E_n)^{-1} \nabla E_n$ 
20:      if  $\boldsymbol{\delta} \cdot \nabla E_n < 0$  then
21:         $\mathbf{d} = \boldsymbol{\delta}$ 
22:      else
23:         $\mathbf{d} = -[diag(H(E_n))]^{-1} \nabla E_n$ 
24:       $\beta = 1$ 
25:      repeat
26:         $(\tilde{x}_{n+1}, \tilde{s}_{n+1}) = (\tilde{x}_n, \tilde{s}_n) + \beta \mathbf{d}$ 
27:         $\beta = \beta / 2$ 
28:      until ZHANGHAGER( $\tilde{x}_{n+1}, \tilde{s}_{n+1}$ ) = true
29:      if any( $sign(\tilde{s}'_{n+1}) \neq sign(\tilde{s}'_0)$ ) then
30:         $(x_{n+1}, s_{n+1}) = (x_n, s_n)$ 
31:         $\mu = \|\tilde{\mathbf{x}}_n - \boldsymbol{\alpha} \circ \tilde{s}_n\|_\sigma^2$ 
32:        logBarrier = true
33:      break
34:    if logBarrier = false then
35:      break
36:  return  $\tilde{\mathbf{x}}_n, \tilde{s}_n$ 

```

---

8), we generate the high-order interpolative meshes. At each element, we set:

$$\tilde{\mathbf{x}}(\xi) = \mathbf{x}^0(\xi) + \sum_{i=2}^p \tilde{\mathbf{x}}_i N_i^p(\xi), \quad (30)$$

$$\tilde{s}(\xi) = s^0(\xi) + \sum_{i=2}^q \mathbf{x}_i N_i^q(\xi), \quad (31)$$

where  $\mathbf{x}^0, s^0$  are fixed (free) when optimizing  $\tilde{E}(E)$ .

We then initialize the log barrier variable (line 9) and enter the optimization loops. The outer loop (starting at line 10) corresponds to the penalized problem (equa-

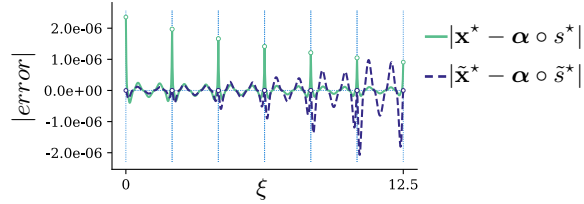
tion (19)). The inner loop (lines 15-34) is the backtracking Newton scheme minimizing  $\tilde{E}(\tilde{\mathbf{x}}^*, \tilde{s}^*)$ . We compute the gradient and Hessian (line 16) and check the stopping criteria (lines 17-19). In our experiments, it corresponds to  $\text{tol} = 10^{-12}$ . Then, a descent direction is chosen (lines 19-23) and the solver enters a loop until the line search condition is met (lines 25-28). Finally, we sample  $s'$  for any changes in its sign. If so, we activate the log barrier (lines 29-32). The main function returns the optimized pair  $(\tilde{\mathbf{x}}_n, \tilde{s}_n)$ .

## 4. EXPLOITING LOCAL HIGHER ORDER ACCURACY

### 4.1 Constrained versus unconstrained optimization

Here we compare the convergence of the solution to the target geometry for the constrained and unconstrained optimization of the disparity. Then, we study the error behaviour focusing on a single element and show that it is possible to attain super-convergence optimizing only the internal nodes.

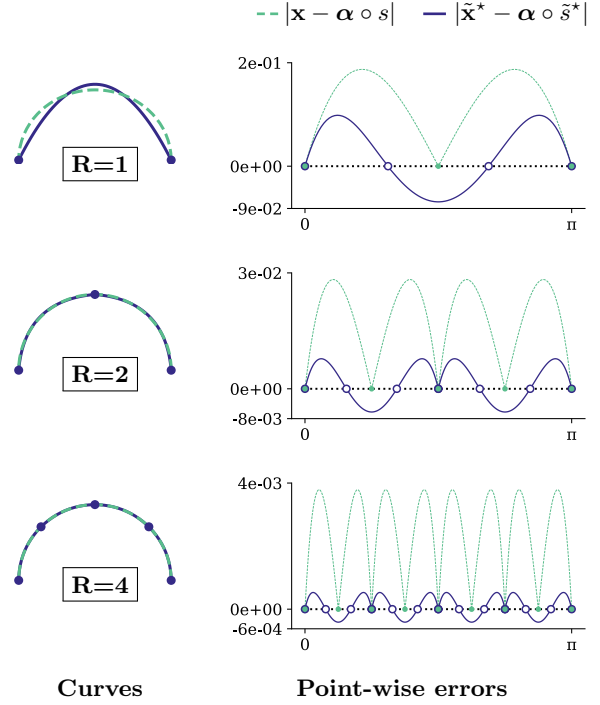
In Figure 9, we show the point-wise errors of the original  $(\mathbf{x}^*, s^*)$  and constrained  $(\tilde{\mathbf{x}}^*, \tilde{s}^*)$  optimization using the spiral from Section 3 as the target geometry (Figure 6). The point-wise error  $|\tilde{\mathbf{x}}^* - \alpha \circ \tilde{s}^*|$  is larger than  $|\mathbf{x}^* - \alpha \circ s^*|$ . Also, unlike  $\mathbf{x}^*$ ,  $\tilde{\mathbf{x}}^*$  interpolates at the element interfaces. We will see that despite having larger errors, the constrained problem preserves the super-convergent behavior from the original disparity.



**Figure 9:** spatial error distribution showing element interfaces (light blue) for the minimization arguments  $(\mathbf{x}^*, s^*)$  (solid green) and  $(\tilde{\mathbf{x}}^*, \tilde{s}^*)$  (dashed deep blue) of the original and constrained disparity, respectively.

Fixing element interfaces transforms the optimization problem in  $R$  (total elements) independent copies. This is illustrated in Figure 10 for a semi-circle successively split into 1, 2 and 4 elements and the corresponding errors after optimizing the constrained disparity. Note that only the internal nodes are considered during optimization, reducing the problem dimension.

In Figure 11, we show convergence plots for a circle and a sphere arc for  $p = 2, 3, 4$  and five mesh refinements. For the 2D case, as for the original disparity, we attain  $2p$  order. For the 3D case, we obtain  $\lfloor \frac{3}{2}(p-1) \rfloor + 2$  order. This means that the  $p = 2$  leads



**Figure 10:**  $h$ -refinement ( $R=1, 2, 4$  elements) for  $p = 2$  meshes. Left: optimized curves  $\tilde{\mathbf{x}}^*$  (solid deep blue) and  $\alpha \circ \tilde{s}^*$  (dashed green). Right: error curves featuring roots (dots) and interface points (solid dots).

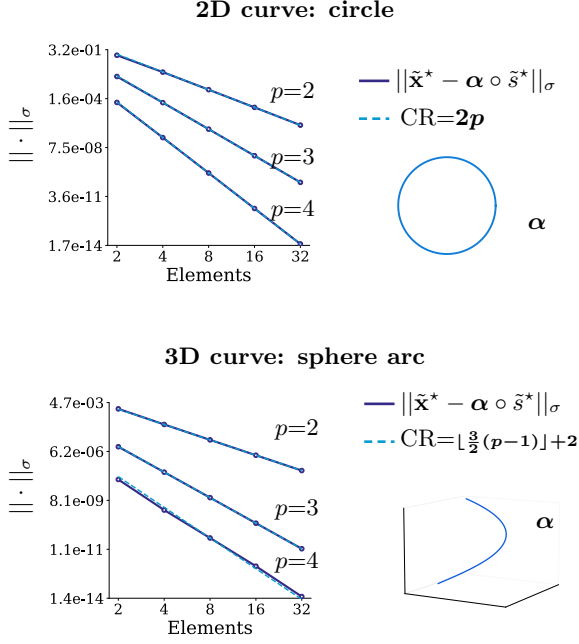
to the usual third order. However, the errors are lower than those resulting from direct interpolation. Later, when we look at the error over a single element, we discuss why we attain respectively,  $2p$  and  $\lfloor \frac{3}{2}(p-1) \rfloor + 2$ .

Now, we use as the target geometry the edges of a CAD model. Figure 12 shows convergence plots for the top edge. Although the disparity values with fixed interfaces are slightly larger, the order of accuracy is the same as freeing interfaces and both cases significantly improve the initial approximation. In Figure 13, we show the point-wise errors after optimizing all edges for  $p = 2, 3$  and 20 elements per edge. Concerning the initial approximation, both the constrained and original optimization decrease the errors with the same magnitude.

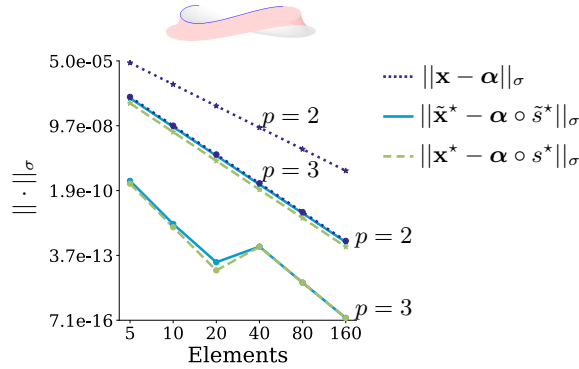
### 4.2 Planar curves: error profile for a single element.

Here, we study the local behaviour of the optimizer focusing on a single element. We use a semi-circle as the target geometry to make the plots clearer.

In Figure 14, we show the point-wise error plots when approximating a semi-circle with a single element for

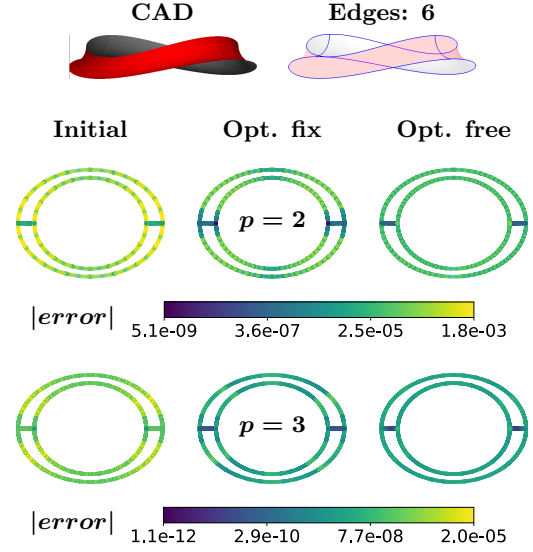


**Figure 11:** slopes (log-log) of the  $\|\cdot\|_\sigma$  norm for several mesh refinements showing the convergence rates (CR) for a 2D (top) and 3D (bottom) after optimizing the constrained disparity ( $\tilde{\mathbf{x}}^*, \tilde{s}^*$ ).

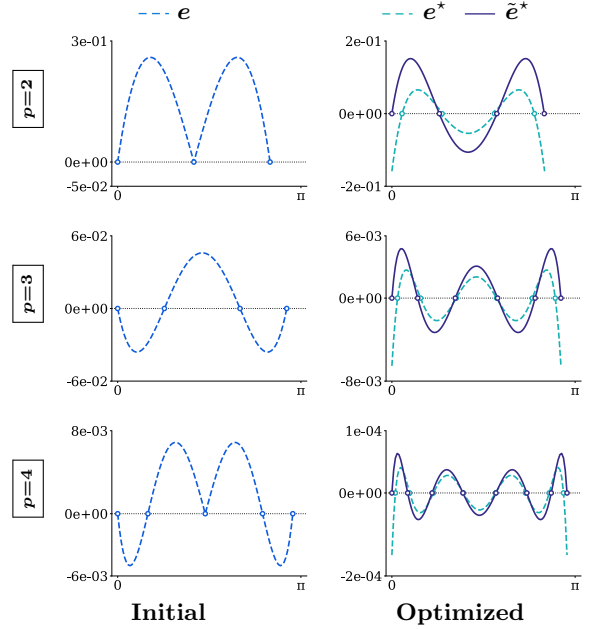


**Figure 12:** slopes (log-log) of the  $\|\cdot\|_\sigma$  norm for several mesh refinements for the top curve of the CAD model (marked in blue) using direct interpolation (dotted dark blue) vs. optimizing the constrained (solid light blue) and the original (dashed green) disparities.

several polynomial degrees. The  $y$ -axis denotes the magnitude of the error  $\mathbf{e} = \mathbf{e}(\xi) = |\mathbf{x}(\xi) - \alpha \circ s(\xi)|$ . The initial approximation (interpolation) is a polynomial of degree  $p$  and the error curve has the expected behaviour:  $p+1$  roots. The right plots show the results from both optimizations: fixed and free element interfaces. Notice that although the fixing interfaces produces slight larger errors, both solutions behave similarly: the curves have  $2p$  roots (instead of  $p+1$ ).



**Figure 13:** point-wise errors  $|\mathbf{x} - \alpha \circ s|$  (top view) approximating the edges of a CAD model with meshes made of 20 elements and  $p=2,3$ , respectively. From left to right: direct interpolation, optimizing the constrained (fix) and original disparities.



**Figure 14:** point-wise error plots  $\mathbf{e} = |\mathbf{x} - \alpha \circ s|$  approximating a semi-circle with a single element before (left) and after optimizing (right) the constrained (solid deep blue,  $(\tilde{\mathbf{x}}^*, \tilde{s}^*)$ ) and the original disparity (dashed light blue,  $(\mathbf{x}^*, s^*)$ ).

In Section 2 (Figure 2) we showed with a circle that the point-wise errors align with the curve normal direction

after optimization. Now we will discuss how this can be related to the disparity super-convergent property. Denote  $\{\mathbf{t}, \mathbf{n}\}$  the curve tangent and normal vectors respectively. For planar curves, we can decompose the error  $\mathbf{e} = \mathbf{x} - \boldsymbol{\alpha} \circ s$  along these directions:

$$\mathbf{e} = (\mathbf{e} \cdot \mathbf{t})\mathbf{t} + (\mathbf{e} \cdot \mathbf{n})\mathbf{n}.$$

The parametric mesh  $s$  uses polynomials of degree  $q$  so at each element, we have a total of  $q + 1$  degrees of freedom. On the other hand, since our physical mesh uses polynomials of degree  $p$  in  $\mathbb{R}^2$ , it has  $2(p + 1)$  degrees of freedom per element. Since our problem is constrained (fixed interfaces) we have a total of  $q + 1 - 2 + 2(p + 1 - 2)$  degrees of freedom (per element). Hence, we can have  $(2p - 2) + (q - 1)$  equations that will be optimizing the disparity.

During optimization, we impose zero tangent error (weakly) in  $q - 1$  equations. If we assume that solving the nonlinear equations behaves similar to interpolation, we would expect at least  $q + 1$  roots in the error function. Recall that the end-points are fixed, hence why we go from  $q - 1$  to  $q + 1$ . This is shown in Figure 12 for the  $q = 2p - 1$  case: the optimized tangent error has 5 and 7 roots for  $p = 2, 3$ , respectively.

The  $2p - 2$  remaining equations are used to impose the total error equal to zero. Assume we can make the tangent error as small as desired by increasing  $q$ . At the optimum, we can think of these  $2p - 2$  equations essentially imposing zero normal error (weakly). With the same reasoning as for  $s$ , we expect  $2p$  roots along the normal component. The  $+2$  corresponds to the interfaces which are interpolation points. This can be appreciated in Figure 15 looking at the plots from the optimized case. Also, provided  $q > 2p - 1$ , the normal error dictates the overall accuracy. Note that in both  $q = 2p - 1$  and  $q = 10$ , the normal error is larger.

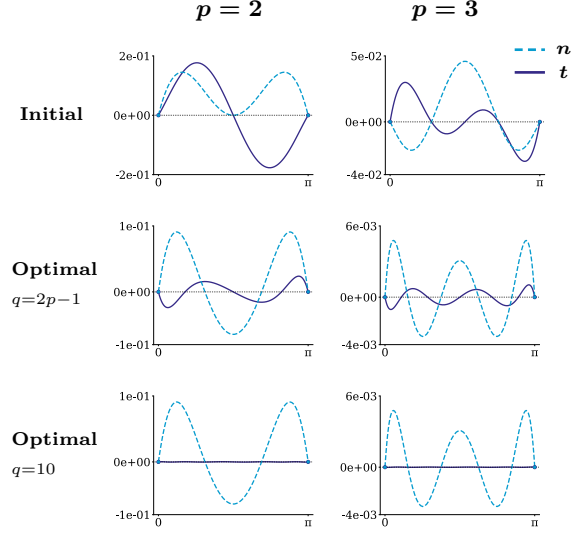
### 4.3 Discussion for 3D curves

We have just discussed the 2D case and how the optimal error behaves in terms of the tangent and normal component. We will now extend our results to the 3D case. In this case, the error decomposition becomes:

$$\mathbf{e} = (\mathbf{e} \cdot \mathbf{t})\mathbf{t} + (\mathbf{e} \cdot \mathbf{n})\mathbf{n} + (\mathbf{e} \cdot \mathbf{b})\mathbf{b}, \quad (32)$$

where  $\mathbf{e} = \mathbf{x} - \boldsymbol{\alpha} \circ s$  and  $\{\mathbf{t}, \mathbf{n}, \mathbf{b}\}$  are the curve tangent, normal and binormal vectors, respectively. Our physical mesh uses polynomials of degree  $p$  in 3D space with fixed end-points. So, at each element, we have  $3(p - 1)$  degrees of freedom. As for the 2D case, we impose in  $q - 1$  equations zero tangent error (weakly). The rest  $3(p - 1)$  equations impose total zero error (weakly).

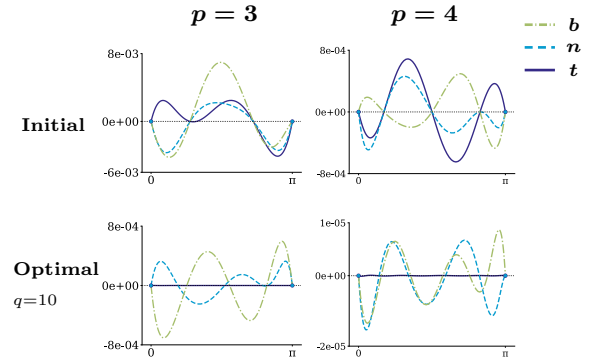
As before, we assume that the tangent error decreases as  $q$  increases. At the optimum, the combined solution



**Figure 15:** tangent ( $\mathbf{t}$ ) and normal ( $\mathbf{n}$ ) error components approximating a semi-circle with one element before and after optimizing the internal nodes.

implies that we have  $3(p - 1)$  equations imposing zero along both the normal and binormal components. In analogy with the 2D discussion, we now expect at least  $\lfloor \frac{3}{2}(p - 1) \rfloor$  interpolation points along each component:  $\{\mathbf{n}, \mathbf{b}\}$ . Since end-points interpolate the curve, it gives  $(\lfloor \frac{3}{2}(p - 1) \rfloor + 2)$  roots per component.

In Figure 16, we show the error plots before and after optimizing the constrained disparity approximating a sphere arc with a single element. As for the 2D case, as  $\tilde{\mathbf{x}}$  follows the image of  $\boldsymbol{\alpha} \circ \tilde{s}$ ,  $\tilde{s}$  minimizes the tangent error. Notice that when  $s$  is of degree  $q = 10$ , the tangent error is negligible compared to the other two components. Also, observe how we obtain both along the normal and binormal directions:  $5 = \lfloor \frac{3}{2}(3 - 1) \rfloor + 2$  roots for  $p = 3$  and at least  $6 = \lfloor \frac{3}{2}(4 - 1) \rfloor + 2$  for  $p = 4$ .

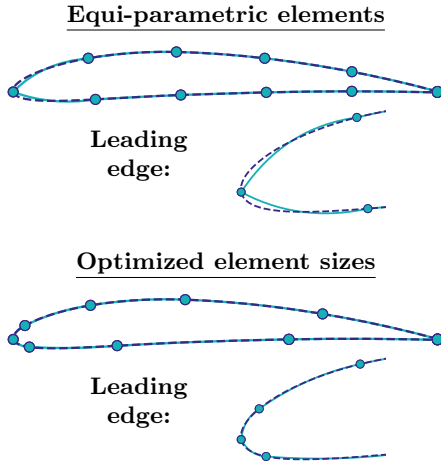


**Figure 16:** tangent, normal and binormal  $\{\mathbf{t}, \mathbf{n}, \mathbf{b}\}$  errors approximating a sphere arc with one element before and after minimizing the constrained disparity.

#### 4.4 Mesh initialization

Here, we focus on the element partition. In Figure 17 (top), we show a  $p = 2$  mesh approximating a spline-based NACA curve with ESP [12] using direct interpolation. In this case, the elements are equi-distributed along the curve parametric space. Notice that the leading edge is poorly resolved. We can improve the element partition optimizing the original disparity using  $p = q = 1$  meshes. Then, we save the partition in  $s^*$  and  $p$ -refine both meshes:  $s$  and  $x$ . The result is shown at the bottom images from Figure 17. Notice that now the leading edge is well approximated. Alternatively, we could have obtained the initial partition performing an arc-length based optimization [16].

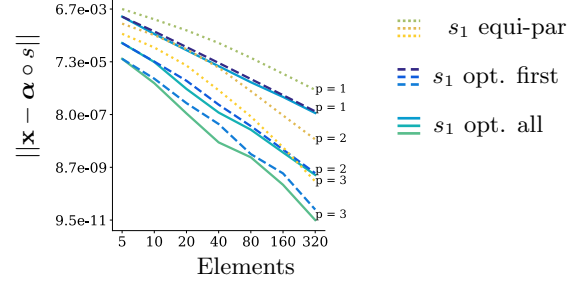
In Figure 18, we study the accuracy of the initial meshes (before optimization) for  $p = 1, 2, 3$  when approximating the NACA curve. We compare sampling directly along the parametric space ( $s_1$  equi-par) with the pre-processing step: optimizing the linear meshes ( $s_1$  opt. all). Notice that the errors significantly improve for the latter one. We also show the case where only the coarser mesh is optimized ( $s_1$  opt. first). Then, the finer meshes are obtained splitting directly each element in two. In this case, the errors are comparable to optimizing at every refinement. This approach can be used to save computational time.



**Figure 17:** interpolating a NACA curve with 10 elements ( $p = 2$ ) using an equi-parametric distribution (top) vs. the proposed pre-processing step (bottom).

### 5. NUMERICAL RESULTS

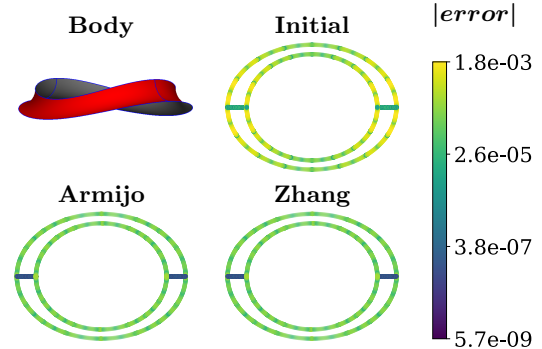
Here, we perform several numerical experiments for the solver performance in terms of the number of iterations. We start studying the impact of the line search choice. Finally, we discuss the trade-offs between errors and iterations comparing the constrained disparity to the original formulation.



**Figure 18:** log-log error plots of the initial meshes approximating a NACA curve for several refinements comparing equi-parametric elements (dotted lines) to optimizing only the coarser mesh (dashed lines) and optimizing at every  $h$ -refinement (solid lines).

#### 5.1 Zhang-Hager vs. Armijo line search

Here, we compare the performance between Armijo and the average line search. In Figure 19, we show the error contours for a CAD body to highlight that both Armijo and Zhang search produce exactly the same solution. In Figure 20, we compare iterations for three different curves and polynomial degrees. Looking at the number of nonlinear iterations, Zhang-Hager line search is systematically faster than Armijo. On average, it is 76% faster (in terms of iterations).



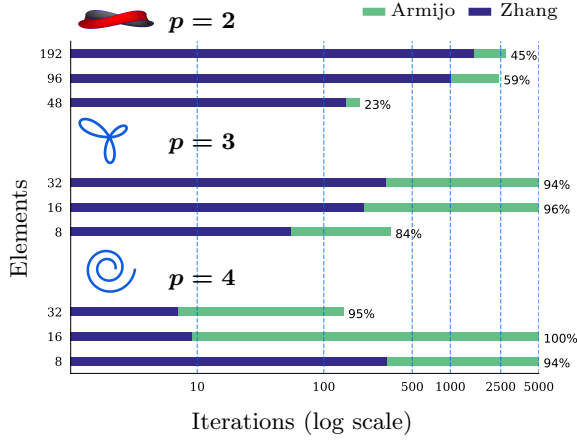
**Figure 19:** point-wise errors (top view) at the six edges of a CAD body before and after optimizing the original disparity using Armijo vs. Zhang-Hager rule. Each edge consists of eight  $p = 2$  elements.

#### 5.2 Constrained optimization: errors vs. iterations

Here, we focus on more complex bodies made of several surfaces and study the trade-offs from fixing the interfaces. We omit straight edges since they can be represented exactly with linear elements and set a stop criterion of  $|\nabla(E)| < 10^{-12}$ .

In Figure 21, we show a CAD model made of 54 edges out of which 36 are curved. We use meshes consist-





**Figure 20:** Armijo vs. Zhang line searches optimizing curves (analytic and CAD) for degrees  $p = 2, 3, 4$ . The % in the bars indicate relative lesser iterations.

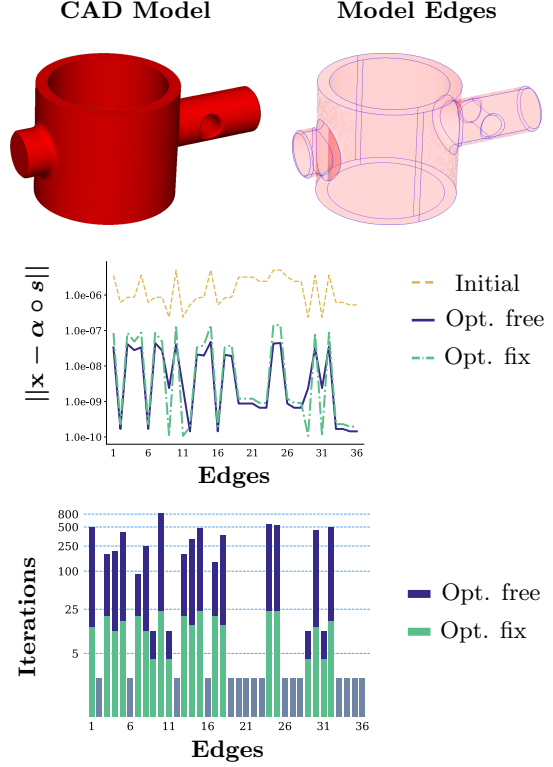
ing of  $p = 3$ ,  $q = 9$  elements and compare the results from computing the constrained and original disparities. The optimal pair  $(\mathbf{x}^*, s^*)$  (free interfaces) produces lower errors, although in some cases, the constrained solution  $(\tilde{\mathbf{x}}^*, \tilde{s}^*)$  leads to a lower disparity value. This is because, in that case, the optimization of the original disparity converged to a local minimum with a higher disparity value. Optimizing the disparity with fixed interfaces took a maximum of 23 iterations whereas the unconstrained problem took a maximum of 808. On average, the constrained problem converges in 9 iterations, and the original in 167.

Finally, in Figure 22 we present an aircraft model consisting of 102 faces and 238 edges. Since the model is symmetric, we study only its left half. This gives 51 curves that we approximate with  $p = 2$  meshes. In this case, optimizing the constrained disparity took a maximum of 43 iterations whereas optimizing the original disparity went, in many cases, beyond 500 iterations. In both cases, we have used the Zhang-Hager line search. On average, the constrained problem takes 4 iterations to converge compared to 387 taken by the unconstrained problem, becoming 87% faster.

## 6. CONCLUSIONS

We have developed a robust solver designed to minimize the disparity measure. We have introduced a log barrier penalty term to avoid curve tangling. The Zhang-Hager average line search is less restrictive, producing the same results as the Armijo rule in significantly fewer iterations. On average, it reduces the number of iterations by 76%.

The original disparity (free interfaces) gives optimal errors. On the other hand, the constrained dispar-



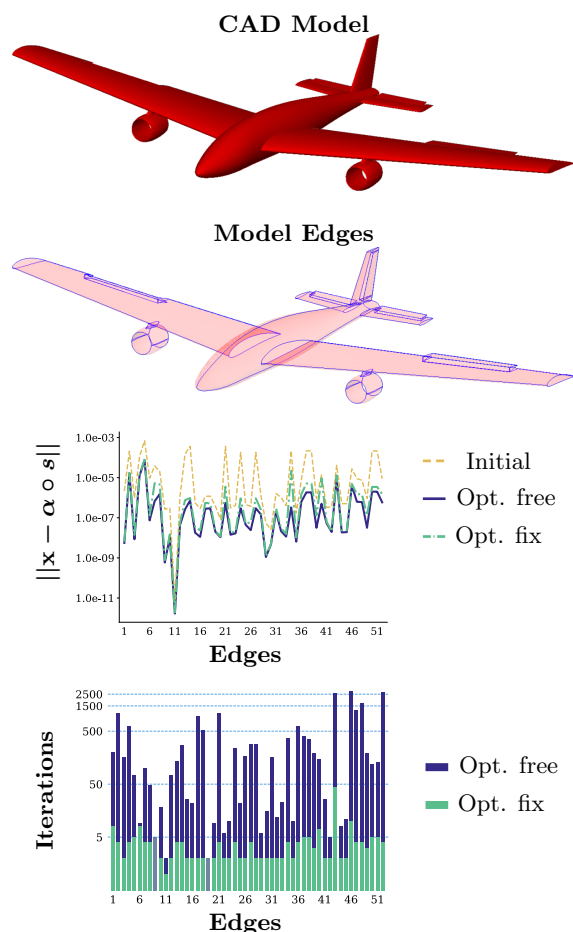
**Figure 21:** fix vs. free interfaces optimization for  $p = 3$  meshes approximating all the curved edges of a CAD model. Total elements: 560.

ity (fixed interfaces) is sub-optimal in terms of the error but still yields super-convergence. We have numerically shown how both disparities are  $2p$  super-convergent for 2D curves and  $\lfloor \frac{3}{2}(p-1) \rfloor + 2$  for curves in 3D space.

Initially, solving the original disparity with the Armijo rule took, on average, around 2000 nonlinear iterations. Our experiments for fixed element interfaces show that optimizing the disparity with the Zhang-Hager line-search, produces a residual less than  $10^{-12}$  in less than 10 iterations. This corresponds to a reduction factor of 100 when compared to the original optimization of the problem. In the future, we will extend this methodology to surface mesh generation.

## 7. ACKNOWLEDGEMENTS

This project has received funding from the European Unions Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 893378 as well as the European Research Council (ERC) grant agreement No 715546.



**Figure 22:** fix vs. free element interfaces optimization for  $p = 2$  meshes approximating all the curved edges of an aircraft model. Total elements: 1375.

## References

- [1] Slotnick J.P., Khodadoust A., Alonso J., Darmofal D., Gropp W., Lurie E., Mavriplis D.J. “CFD vision 2030 study: a path to revolutionary computational aerosciences.” Tech. Rep. NASA/CR-2014-218178, 2014
- [2] Alt H., Godau M. “Computing the Fréchet distance between two polygonal curves.” *Int. J. Comput. Geom. Appl.*, vol. 5, 75–91, 1995
- [3] Remacle J., Lambrechts J., Geuzaine C., Toulorge T. “Optimizing the geometrical accuracy of 2D curvilinear meshes.” *Procedia Eng.*, vol. 82, 228–239, 2014. 23rd International Meshing Roundtable
- [4] Toulorge T., Lambrechts J., Remacle J.F. “Optimizing the geometrical accuracy of curvilinear meshes.” *Journal of Computational Physics*, vol. 310, 361–380, 2016
- [5] Ruiz-Girons E., Sarrate J., Roca X. “Defining an  $\mathcal{L}_2$ -disparity measure to check and improve the geometric accuracy of non-interpolating curved high-order meshes.” *Procedia Eng.*, vol. 124, 122–134, 2015. 24th International Meshing Roundtable
- [6] Ruiz-Girons E., Sarrate J., Roca X. “Generation of curved high-order meshes with optimal quality and geometric accuracy.” *Procedia Eng.*, vol. 163, 315–327, 2016. 25th International Meshing Roundtable
- [7] Ruiz-Girons E., Sarrate J., Roca X. “Measuring and improving the geometric accuracy of piecewise polynomial boundary meshes.” *J. Comput. Phys.*, vol. 443, 110500, 2021
- [8] Dai Y.H. “On the nonmonotone line search.” *Journal of Optimization Theory and Applications*, vol. 112, no. 2, 315–330, 2002
- [9] Grippo L., Lampariello F., Lucidi S. “A nonmonotone line search technique for Newtons method.” *SIAM Journal on Numerical Analysis*, vol. 23, no. 4, 707–716, 1986
- [10] Zhang H., Hager W.W. “A nonmonotone line search technique and its application to unconstrained optimization.” *SIAM Journal on Optimization*, vol. 14, no. 4, 1043–1056, 2004
- [11] Toint P.L. “An Assessment of nonmonotone linesearch techniques for unconstrained optimization.” *SIAM Journal on Scientific Computing*, vol. 17, no. 3, 725–739, 1996
- [12] Haimes R., Dannenhoffer J. “The engineering sketch pad: A solid-modeling, feature-based, web-enabled system for building parametric geometry.” Sep. 2013. 21st AIAA Computational Fluid Dynamics Conference
- [13] Garimella R.V., Shashkov M.J., Knupp P.M. “Triangular and quadrilateral surface mesh quality optimization using local parametrization.” *Computer Methods in Applied Mechanics and Engineering*, vol. 193, no. 9, 913–928, 2004
- [14] Dobrev V., Knupp P., Kolev T., Mittal K., Tomov V. “The Target-Matrix Optimization Paradigm for High-Order Meshes.” *SIAM Journal on Scientific Computing*, vol. 41, no. 1, B50–B68, 2019
- [15] Nocedal J., Wright S. *Numerical Optimization*. Springer Science & Business Media, 2006
- [16] McLaurin D., Shontz S.M. “Automated edge grid generation based on arc-length optimization.” *Proceedings of the 22nd International Meshing Roundtable*, pp. 385–403. Springer International Publishing, 2014



# AUTOMATIC GENERATION OF LOAD-BALANCING-AWARE BLOCK-STRUCTURED GRIDS FOR COMPLEX OCEAN DOMAINS

Daniel Zint<sup>1,2</sup>      Roberto Grosso<sup>1</sup>      Vadym Aizinger<sup>3</sup>      Sara Faghih-Naini<sup>1,3</sup>  
Sebastian Kuckuk<sup>1,4</sup>      Harald Köstler<sup>1</sup>

<sup>1</sup>*Friedrich-Alexander-University Erlangen-Nuremberg, Germany, daniel.zint@fau.de*

<sup>2</sup>*INRIA Sophia-Antipolis, France, daniel.zint@inria.fr*

<sup>3</sup>*University of Bayreuth, Germany*

<sup>4</sup>*Erlangen National High Performance Computing Center (NHR@FAU)*

## ABSTRACT

Many high-performance computing applications involve sophisticated finite element simulations on complex domains and, for this reason, often cannot use a single structured grid for the entire domain. A popular alternative are block-structured grids (BSGs) that are more flexible geometrically but still offer a significant amount of structure. However, the standard generation process for BSGs relies heavily on manual input to define the segmentation of the computational domain – a rather difficult task to perform for complex geometries. Ocean domains often contain fractal boundary shapes and details such as islands and channels that cannot be accurately represented using BSGs. We present a method to automatically generate BSGs with an exactly specified number of blocks for real-world domains arising in 2D ocean simulations. Our BSGs consist of quad blocks refined via structured triangular grids and employ masks to accurately represent small features. The performance of the proposed BSG generation method is evaluated for realistic ocean domains and validated using simulations of the two-dimensional shallow water equations discretized by the discontinuous Galerkin method.

**Keywords:** mesh generation, block-structured grids, high-performance computing, discontinuous Galerkin method, ocean simulation, shallow water equations

## 1. INTRODUCTION

The accuracy and the computational performance of finite element models is strongly affected by the type and the quality of the employed computational mesh. Structured grids enable memory access in repeated regular stencils and therefore offer nearly optimal efficiency [1]. Discretizations utilizing unstructured meshes need to additionally load indexing data and they access memory in irregular fashion resulting in cache misses which reduce performance [2]. Nevertheless, unstructured meshes are often favored due to their geometrical flexibility – many domains with complex boundaries and varying element sizes cannot be

accurately represented by structured grids at all – and the ability to adapt resolution in accordance with the application requirements. Furthermore, the grid structure also plays an important role for load balance in distributed computations. As pointed out in [3], the grid should be adapted to the hardware that is used for the simulation. In addition, the size of mesh elements determines the maximum admissible time step in explicit simulations of time-dependent problems, whereas the element shape critically affects the stability of a finite element discretization. A triangular element, for example, is considered optimal if it is equilateral; the more it is distorted the worse its quality. Element quality can be measured by the mean ratio

metric [4, 5],

$$q_{mrm} = 4\sqrt{3} \frac{A}{\sum_{i=0}^3 l_i^2}, \quad (1)$$

where  $A$  is the signed area of the triangle (the sign indicates flipped triangles), and  $l_i$  are the lengths of its edges. Numerical errors caused by low-quality elements degrade the results or may even cause a blow-up of the simulation. Also, the element size has an influence on the discretization error.

A compromise between the performance of a structured grid and the flexibility of an unstructured one is a block-structured grid (BSG). It consists of an unstructured block-mesh where each block contains a structured grid. BSGs certainly alleviate the problem with complex domains but do not solve it completely. Since BSGs are complicated to generate automatically they are mostly used for simple domains, and the block structure is usually optimized for specific applications (e.g. turbine blades). For ocean domains, no such simple segmentation is possible; in addition, real-world geometries often contain application-critical small-scale features. This presents a major difficulty for the BSG methodology: Given a certain minimum block size, islands or other domain features smaller than this size cannot be represented.

In the current work, we introduce and evaluate a masking approach aiming to solve this issue: Our BSGs are generated for simplified geometries that do not resolve features smaller than the given block size; instead, the excessive elements (those outside of the correctly resolved geometry) are masked, i.e. excluded from the simulation. Starting from a user-provided unstructured triangular mesh, our method automatically generates a BSG of a given density with a prescribed number of topologically uniform blocks. Optimal load balance in a parallel simulation is achieved by choosing the number of blocks to be a multiple of the processing units. Complex boundaries and small islands that usually cannot be represented with BSGs are restored by masking elements and repositioning boundary vertices. The code is available at <https://github.com/DanielZint/hpmeshgen>.

This paper is structured as follows. In Section 2, we describe related work. The generation of the block structure is presented in Section 3. The refinement of blocks, the masking, and the adaptation to the domain are described in Section 4. BSGs for selected real-world domains and simulation results used for validation of our approach can be found in Section 5, and a short Conclusions & Outlook section wraps up this work.

## 2. RELATED WORK

Considering that BSGs are used in many high-performance computing applications, e.g. [6, 7, 8, 9], there is surprisingly little literature on generating such grids. Armstrong et al. showed in [10] that methods for generating BSGs share the same difficulty, namely the placement of mesh singularities. Fogg et al. use cross-fields for generating a block structure [11, 12]. Lim et al. propose an evolutionary algorithm for block generation [13, 14]. Sánchez and Cruz present a semi-automatic approach for parametrizing polygonal regions [15], in which a polygonal region is decomposed into quadrilateral blocks and refined via structured quad grids. By enabling manual correction of the decomposition, the blocks are large and results look promising. A similar approach was presented in [16]. However, these methods focus on rather simple domains which are decomposed into a small number of blocks. The ocean domains in the focus of the current study are much more complex geometrically, and we have additional constraints such as the exactly prescribed number of blocks and the CFL condition, Equation (2).

A method for generating BSGs for complex ocean domains was presented in [17]. It takes into account the CFL condition, but its performance is limited by the domain geometry: Realistic coastal regions with fractal shapes and small islands cannot be represented accurately. In addition, the method in [17] does not produce an exact number of blocks and therefore may cause load imbalances. Nevertheless, our current scheme follows the same idea of generating blocks by simplifying an unstructured triangular mesh.

BSG generation also appears in geometry processing where blocks are used for efficiently storing textures and the grid itself. Boier-Martin et al. [18] and Carr et al. [19] use clustering techniques for block creation. Dong et al. [20] quadrangulate any manifold by applying a Morse-theoretic analysis to the eigenvectors of the mesh Laplacian. Daniels et al. present an algorithm for quadrilateral remeshing [21]. It requires closed manifold meshes and is therefore not transferable to 2D ocean meshes. None of the above methods considers element quality as the meshes are not designed for numerical simulations.

Campan [22] presented a survey of methods for partitioning surfaces into quadrilateral patches. The methods presented there have a different objective. Blocks do not have a prescribed size, and the number of blocks is also not fixed. Therefore, these methods are not appropriate for HPC.

### 3. GENERATION OF BLOCK STRUCTURE

To generate a quad block structure with a prescribed number of blocks, we first simplify the unstructured triangular mesh, Figure 1a, to twice the prescribed number of blocks, Figure 1b. The coarse triangles are then merged into quads to form a quad block structure, Figure 1c. The quad blocks are refined with structured triangular grids, Figure 1d. Elements that are outside the domain are masked, Figure 1e. Finally, boundary vertices are mapped to the original contour, and element size is restored, Figure 1f.

Quad blocks with structured triangular grids offer advantages and disadvantages: On the one hand, one needs only half as many quad blocks as triangular ones, and the communication topology between quad blocks is simpler to optimize; on the other, it is easier to produce an accurate representation of complex boundaries by masking triangular grids. In addition, the generation process for triangular meshes (used for partitioning into blocks) is robust and produces high-quality partitions, whereas robustly generating unstructured partitions into quads without degenerated elements is a much more complex task. A triangular partition can be converted into a quad one by combining triangles [23, 24]. The resulting quad mesh might have degenerated quads, e.g. the two quads in the top of Figure 1c, but the triangles inside the quad blocks are still valid, Figure 1d. Furthermore, our grid generator was developed for the ExaStencils [25] code generation framework with its python front-end GHODDESS [26] currently limited to quad-type communication topologies.

Aside from complex boundary regions, also the element size must be considered in the mesh generation process. The CFL condition for shallow water equations contains a quotient that describes the relation between element size  $\Delta x$  and ocean depth  $H$  [27],

$$c_m = \frac{\Delta x}{\sqrt{H}}. \quad (2)$$

The largest possible time step is proportional to  $c_m$ ,

$$\Delta t_{\max} \sim c_m = \frac{\Delta x}{\sqrt{H}}. \quad (3)$$

Large elements allow large time steps and therefore faster computation, but the simulation also becomes less accurate. Thus, a compromise has to be found between time step size and accuracy. The element with the smallest  $c_m$  determines the maximum time step  $\Delta t_{\max}$ , whereas the discretization error (and thus the accuracy) is largely controlled by  $\Delta x$ . Therefore,  $c_m$  should be approximately the same for each element, whereas the local mesh resolution should be chosen to provide sufficient numerical accuracy.

### 3.1 Simplification

Our method for simplifying the triangular mesh is based on the ideas of [17]. However, we use a different error metric and vertex positioning method.

Quadric mesh simplification modifies a triangular mesh by performing edge collapses using the quadric error metric [28]. First, the error of all edge collapses is computed. Simplification is then performed iteratively starting with the collapse that causes the smallest error. After each edge collapse, the error of the surrounding edges is recomputed. The simplification terminates when the desired number of triangles is reached or when no more edges can be collapsed.

#### Error Metric

**Definition 1** The relative distance  $\tilde{d}(\mathbf{p}_i, \mathbf{p}_j)$  between two points  $\mathbf{p}_i$  and  $\mathbf{p}_j$  is the quotient of the Euclidean distance  $\|\mathbf{p}_j - \mathbf{p}_i\|$  and the integral of the size function  $h(\mathbf{x})$  between the two points divided by the Euclidean distance:

$$\tilde{d}(\mathbf{p}_i, \mathbf{p}_j) = \frac{\|\mathbf{p}_j - \mathbf{p}_i\|}{\frac{\int_{\mathbf{p}_i}^{\mathbf{p}_j} h(\mathbf{x}) d\mathbf{x}}{\|\mathbf{p}_j - \mathbf{p}_i\|}} = \frac{\|\mathbf{p}_j - \mathbf{p}_i\|^2}{\int_{\mathbf{p}_i}^{\mathbf{p}_j} h(\mathbf{x}) d\mathbf{x}}. \quad (4)$$

A detailed explanation on generating the size function  $h(\mathbf{x})$  from a triangular mesh is given in [17].

**Definition 2** The relative length  $r_e$  of an edge  $e$  is the relative distance of its incident vertices  $v_i$  and  $v_j$  with positions  $\mathbf{p}_i$  and  $\mathbf{p}_j$ ,

$$r_e = \tilde{d}(\mathbf{p}_i, \mathbf{p}_j). \quad (5)$$

**Definition 3** The position error  $\rho_v(\mathbf{x})$  of a vertex  $v$  is the squared relative distance between its initial position  $\mathbf{p}$  and its current position  $\mathbf{x}$ ,

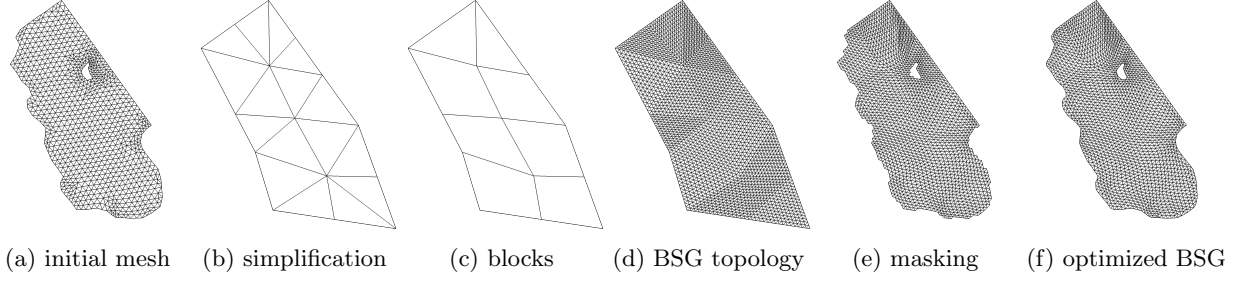
$$\rho_v(\mathbf{x}) = \tilde{d}(\mathbf{p}, \mathbf{x})^2. \quad (6)$$

We derive the simplification error from the position error  $\rho_v(\mathbf{x})$ . To ensure that the mesh is simplified uniformly we keep track of all vertices that were collapsed into a single vertex  $v$  by storing them in a set  $V_c(v)$ . The simplification error  $Q_v(\mathbf{x})$  of vertex  $v$  at position  $\mathbf{x}$  is the sum of all position errors of the vertices that were collapsed into  $v$

$$Q_v(\mathbf{x}) = \sum_{v_c \in V_c(v)} \rho_{v_c}(\mathbf{x}). \quad (7)$$

The error of an edge collapse is the sum of the simplification errors of its vertices  $v_i$  and  $v_j$ ,

$$Q_e = Q_{v_i} + Q_{v_j}. \quad (8)$$



**Figure 1:** The BSG generation steps for the Bahamas domain.

When collapsing  $v_j$  into  $v_i$ , the set  $V_c(v_j)$  is appended to  $V_c(v_i)$ .

Several conditions have to be met for an edge before it can be collapsed. An edge collapse is considered *invalid* if

- the collapsing edge connects two boundary vertices but itself is in the interior, or
- the resulting elements have poor quality.

The first condition ensures that the mesh is not cut open. The second condition prohibits flipped and deformed triangles. We consider a triangle as low quality if the mean ratio metric, Equation (1), is below 0.1. Similar constraints, called link conditions, were formulated by Dey et al. [29].

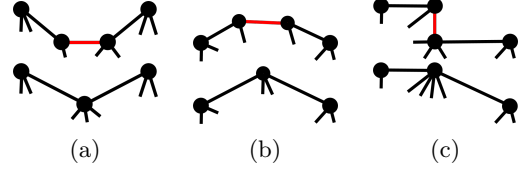
Once an interior boundary (usually an island) only contains three edges, it is replaced by a triangle. Thus, islands disappear if they are too small to be represented in the block structure. Additionally, we make use of non-edge contractions introduced in [30]. Vertices that are close but not connected by an edge are collapsed along a virtual edge.

### Vertex Positioning

The computation of vertex positions after an edge collapse differs for interior and boundary vertices. If both vertices of the collapsed edge are in the interior or the edge is virtual, the remaining vertex is positioned in the middle. More elaborate approaches like optimizing the new vertex position were tested but did not have any significant impact on quality. If an edge connects an interior with a boundary vertex, the edge is collapsed towards the boundary. This ensures that the domain shape remains unchanged.

For boundary edges, we consider several cases depending on the local convexity of the boundary at the two vertices. If the boundary is concave at both vertices, the new vertex is positioned at the midpoint of the

edge, Figure 2a. If both vertices are convex we compute the intersection point of the neighboring edges and use it as the new vertex position, Figure 2b. If one vertex is convex and the other is concave, the convex vertex position is preserved, Figure 2c. This only leads to a valid solution if these edges are not parallel. Otherwise, the edge collapse is considered invalid. The positioning of boundary vertices ensures that the initial mesh is fully covered by the simplified mesh.



**Figure 2:** Vertex positioning at boundaries. The collapsed edge is marked in red.

### 3.2 Remeshing

The simplified mesh is improved with standard post-processing steps like smoothing and edge flipping. We perform remeshing similarly to [31]. First, we compute the average of the relative edge lengths  $\bar{r}_e$ . Then, the mesh is iteratively improved with the following steps:

1. Split all edges whose relative length is larger than  $\frac{4}{3}\bar{r}_e$ .
2. Collapse all edges whose relative length is smaller than  $\frac{3}{4}\bar{r}_e$ . Collapses are executed as described in Section 3.1.
3. Flip edges whenever it reduces the number of irregular vertices. A vertex is considered irregular if it has a valence unequal to 6. An edge with the vertices  $v_1, v_2$  and the incident triangles

$(v_1, v_2, v_3), (v_2, v_1, v_4)$  is flipped if  $e'_\eta < e_\eta$ , where

$$\begin{aligned} e_\eta &= \max\{|\eta_1 - 6|, |\eta_2 - 6|\} \\ &\quad + \max\{|\eta_3 - 6|, |\eta_4 - 6|\}, \\ e'_\eta &= \max\{|\eta_1 - 7|, |\eta_2 - 7|\} \\ &\quad + \max\{|\eta_3 - 5|, |\eta_4 - 5|\}, \end{aligned}$$

and  $\eta_i$  is the *valence* of vertex  $v_i$  (i.e. the number of vertices connected to  $v_i$  by an edge). The valence of a boundary vertex  $v_b$  is increased depending on the boundary angle  $\alpha_b$ ,

$$\eta_b \leftarrow \eta_b + \text{floor}\left(\frac{2\pi - \alpha_b}{\frac{1}{3}\pi}\right).$$

Flips that cause tangling, i.e. triangles with negative quality, Equation (1), are discarded.

4. Improve mesh quality by smoothing. We use the DMO (Discrete Mesh Optimization) approach from [32] with a vertex metric that combines the mean ratio metric, Equation (1), with Laplace smoothing

$$q_{iso}^{(v)}(\mathbf{x}_k) = \begin{cases} q_{mrm}^{(v)}(\mathbf{x}_k), & \text{if } q_{mrm}^{(v)}(\mathbf{x}_k) < 0.5 \\ 0.5 + q_{lap}^{(v)}(\mathbf{x}_k), & \text{otherwise} \end{cases}$$

$$q_{lap}^{(v)}(\mathbf{x}_k) = \frac{1}{\|\mathbf{lp}_k - \mathbf{x}_k\|^2 + 1},$$

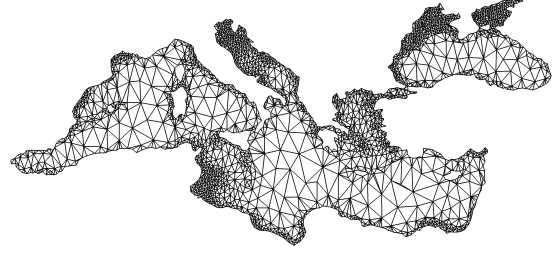
where  $\mathbf{x}_k$  is the position of vertex  $v_k$ , and  $\mathbf{lp}_k$  denotes the center of gravity of the one-ring neighborhood (vertices connected to  $v_k$  by an edge). The advantage of this metric over the pure mean ratio is that the results are smoother, and elements tend to be locally of similar size. The mean ratio metric can cause distortions if mesh topology is not adequate for the domain.

Remeshing is stopped if either the number of triangles does not change within two iterations or if a certain maximum number of iterations is reached (currently, we perform a maximum of 100 iterations). After that, the number of triangles is not generally equal to the number of blocks as prescribed by the user; thus, the last remeshing iteration is carried out to produce the exact number of blocks: We force edge splits if we do not have enough triangles or we collapse them if there are too many – independently of  $\tilde{r}_e$ .

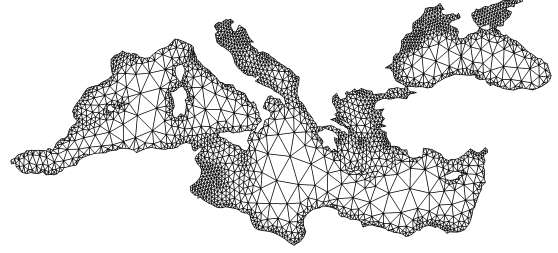
An example showing the effectiveness of remeshing is given in Figure 3. Low quality triangles and vertices with high valence are removed by remeshing, improving the overall mesh quality. Both, the simplified and the remeshed mesh consist of 4000 triangles.

### 3.3 Conversion to Quad Blocks

The triangles in the simplified mesh are merged using Blossom-Quad [23] which relies on Edmonds' Algorithm to find a perfect match for the dual graph of



(a) Simplified



(b) Remeshed

**Figure 3:** Remeshing for domain Mediterranean with 4000 triangles.

the triangular mesh. Unfortunately, not every graph has a perfect match and therefore some triangles might be left over. These triangles always appear pair-wise along mesh boundaries. The solution proposed in [23] duplicates the vertex between the two triangles. This solution is not feasible for us because this increases the number of quads. Instead, we perform triangle merging as proposed several times in literature [24, 33, 34]. This method moves one triangle towards another by flipping edges until they can be merged into a quad.

The mesh must have an even number of triangles for applying Blossom-Quad and triangle merging. The simplification and remeshing might cause an uneven number though. In that case, the relatively longest boundary edge according to Equation (5) is split in two increasing the total number of triangles by one.

Finally, the quad blocks are refined with structured triangle meshes, giving the desired block-structured topology. There are two possible orientations for a triangle mesh within a quad block. We choose the orientation which results in triangles of higher quality, measured with the mean ratio metric, Equation (1).

## 4. GRID ADAPTATION TO DOMAIN

The grid adaptation consists of three main steps. First, element size is adjusted by repositioning interior vertices. Second, the domain shape is restored by masking triangles. Third, boundary vertices are mapped onto the domain shape. Finally, interior vertices are repositioned once more to improve quality

near boundaries.

#### 4.1 Repositioning Interior Vertices

For finding optimal vertex positions in the interior of the BSG, we define a quality metric and optimize vertex positions with DMO [32]. We use the relative edge length defined in Section 3.1 to adapt mesh density. Additionally, we set a minimal mean ratio quality  $\hat{q}_{mrm}^{(v)}$  to ensure numerical stability:

$$q_{d,mrm}^{(v)}(\mathbf{x}) = \begin{cases} q_{mrm}^{(v)}(\mathbf{x}) & \text{if } q_{mrm}^{(v)}(\mathbf{x}) \leq \hat{q}_{mrm}^{(v)} \\ \hat{q}_{mrm}^{(v)} + q_d^{(v)}(\mathbf{x}) & \text{otherwise.} \end{cases}$$

The density quality  $q_d^{(v)}(\mathbf{x})$  of vertex  $v$  at position  $\mathbf{x}$  depends on the longest and shortest relative length of its incident edges:

$$q_d^{(v)}(\mathbf{x}) = \frac{1}{r_{e,max} - r_{e,min} + 1}.$$

Optimizing for density quality sometimes generates wiggly lines. We remove them by applying one iteration of DMO with the mean ratio metric.

#### 4.2 Masking

We trim the mesh such that it represents the domain well by masking elements that are outside of the domain. For this, a signed distance function is utilized to decide which vertices and edges give the best representation of the boundaries. In the first sweep, we mask all triangles that lie outside the domain. This is determined by checking the signed distance of all incident vertices and the center point of the triangle. If all distances are positive, the triangle is masked. Considering the center point of the triangle prevents masking those triangles that have all vertices on the boundary but should not be masked. Due to round-off effects, boundary vertices may have positive signed distances even though they lie on the contour.

In contrast to triangles that must be preserved, it might also happen that triangles lie almost completely outside the domain. For masking those triangles we approximate the area fraction of the triangle that is outside of the domain. If its major part (currently we use a threshold of 85 %) lies outside, it is masked.

Next, we consider boundary vertices. We mask a vertex and its incident triangles if the boundary is approximated better by the vertices on its one-ring. For that we compare the distance of the vertex with all its neighbors. If all neighbors are closer to the contour than the vertex, it is masked.

One more special case must be accounted for before mapping the boundary to the contour. Trimming might cause a zigzag line at the boundary. If this line

is mapped to the contour, triangles might degenerate. We avoid this by masking triangles that would be of low quality when mapped to the contour.

#### 4.3 Boundary Adaptation

Producing non-degenerate triangles is more important than placing boundary vertices perfectly on the domain boundary. Thus, vertices are only mapped onto the contour if the resulting triangles are not degenerated. Fortunately, this happens only rarely. Vertices are mapped by moving them to the closest point on the contour. Afterwards, all boundary vertices are smoothed by positioning them in the midpoint between their neighbors.

### 5. RESULTS

In this section, we compare BSGs to the initial unstructured triangular meshes. Element quality is measured with the mean ratio metric, Equation 1, element size is evaluated by the CFL quotient, Equation 2. For the BSG generation, we use a workstation with an Intel i7-6700k CPU with 4 cores and 4.0 GHz and an NVIDIA GeForce GTX 1070 GPU. Vertex repositioning is performed on GPU everything else on CPU. All runtime measurements cover the whole generation process including read and write operations.

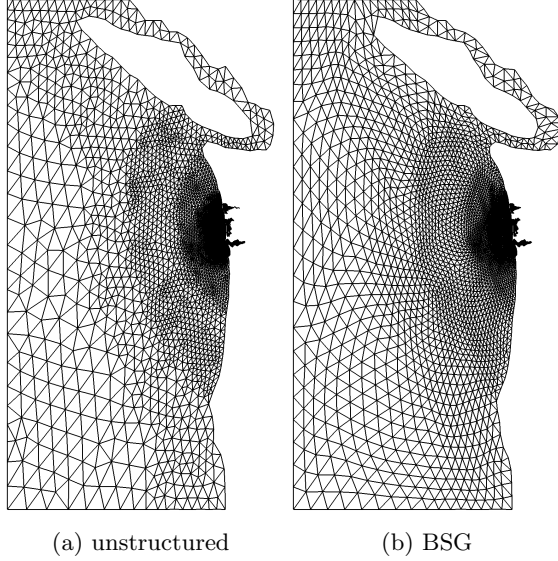
The number of triangles  $|T|$  in a BSG is the product of the number of blocks  $N_B$  and the number of elements per block  $U$ . The structured triangular grid is generated by refining each block uniformly. The number of unmasked triangles is denoted by  $|F|$  and the relative amount of masked triangles by  $\mu$

$$\mu = (|T| - |F|)/|T|. \quad (9)$$

#### 5.1 BSG generation for real-world ocean domains

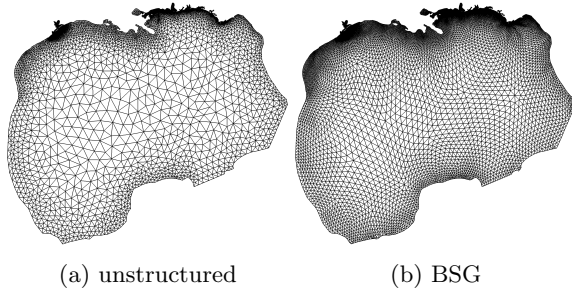
The first example is called Graysharbor, Figure 4a, and represents the geometry and topography of the Grays Harbor in the State of Washington (USA). The element size varies strongly throughout the domain. We generate a BSG with  $N_B = 250$ ,  $U = 128$ , and  $|F| = 27898$ . The generation of the BSG took about 15 seconds. The BSG for Graysharbor contains 13% masked triangles. The minimal mean ratio quality is 0.32. The CFL quotient varies between 83 and 4262. It is similar to the range of the unstructured mesh that is between 81 and 5547. The domain is represented correctly by the BSG, Figure 4b.

The second example is an unstructured mesh representing the Gulf of Mexico with 14269 triangles, Figure 5a. The generation of a BSG with  $N_B = 300$ ,  $U = 128$ , and  $|F| = 32635$  took 8 seconds, Figure 5b.



**Figure 4:** Graysharbor with 34 406 triangles in the unstructured mesh [35] and 27 898 in the BSG.

The minimal mean ratio quality for the BSG is 0.30. Although the BSG has more than double the number of elements, the CFL quotient is very similar for both grids. The unstructured mesh has a CFL quotient between 36 and 11580 and the BSG between 34 and 7304. In this BSG, 15% of the elements were masked.



**Figure 5:** The Gulf of Mexico with 14 269 triangles in the unstructured mesh and 32 635 in the BSG.

## 5.2 Validation test: Mediterranean

The domain Mediterranean is our most complex example; it contains fine-scale geometry features such as small islands and channels which are only one element wide, Figure 6. The unstructured mesh consists of 112 962 triangles, and we generate BSGs with up to 8000 blocks and a higher resolution than the initial unstructured mesh in order to represent details like the Bosphorus connecting the Black Sea with the Marmara Sea correctly. Due to the increased number of elements, the CFL quotient is smaller for BSGs than for

the unstructured mesh. For  $N_B = 4000$  with 472 931 triangles, we have a minimal value of  $c_m = 35.1$ . The mean ratio metric has its minimal value at around 0.3 for all meshes. BSG generation for the Mediterranean domain takes between 103 and 141 seconds.

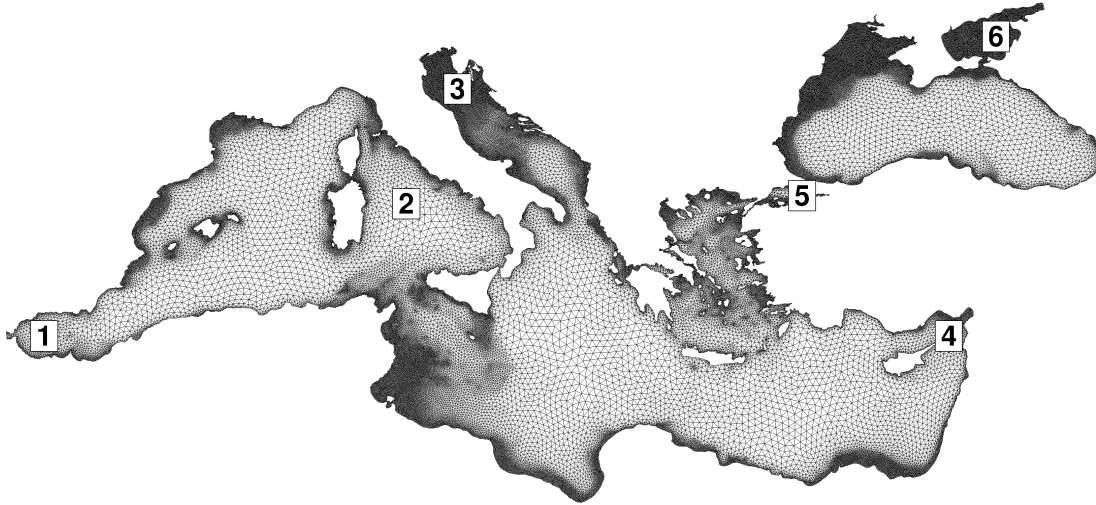
For this domain, we clearly see the advantage of masking elements. Even small details can be captured by BSGs with masks. In Figure 7 we illustrate some details of the BSG with  $N_B = 8000$ . Regions such as the Bosphorus, where the mesh must be connected, are represented correctly, Figure 7a. In other regions such as the Gulf of Corinth, the isthmus is also shown correctly in the BSG, Figure 7d. The small land bridge connecting southern Greece with the Peloponnese cannot be represented on the block level but is restored by masking in the refined mesh. Sicily, on the other hand, is correctly shown as disconnected from the mainland of Italy, Figure 7h. Coastal regions are represented substantially better by the refined and masked grid than by the unmasked block grid, Figure 7f. Most of these details could not be represented without masking as triangles would be highly distorted, e.g. imagine fitting whole blocks into the Bosphorus, Figure 7a.

$N_B$	$U$	$ F $	$\mu$	runtime/s
8 000	32	244 295	5 %	103
4 000	128	472 931	8 %	141
2 000	128	226 350	12 %	137

**Table 1:** BSG configurations for Mediterranean with masks, where  $N_B$ ,  $U$ , and  $|F|$  are the number of blocks, elements per block, and unmasked elements respectively.  $\mu$  is the relative amount of masked elements.

To validate the quality of the generated BSGs we additionally simulate a circulation scenario for the Mediterranean and compare results between the initial unstructured mesh and various generated BSGs. The simulations are performed using the 2D shallow water equations solver UTBEST [27] based on the discontinuous Galerkin (DG) method. The tide-driven flow is simulated for 5 days starting from the cold start conditions (zero elevation and velocity) and uses piecewise constant DG discretization combined with the forward Euler time stepping. Since our BSGs for this test case have somewhat higher resolution, the used time step is between 4 and 6 seconds compared to the maximum time step of 9 seconds for the original unstructured mesh. The free surface elevation written out at 6 locations (recording stations), Figure 8, is compared to the results produced by the unstructured mesh simulation. For all stations – even at Station 6 in the Sea of Azov and thus the farthest from the open boundary located in the Straits of Gibraltar – the results match well, with the largest difference at Station 3 in the Adriatic Sea. We attribute this difference to a higher





**Figure 6:** Unstructured mesh for domain Mediterranean with 112 962 triangles.

resolution of our BSGs.

The relative number of masked triangles, Equation (9), is low in all presented BSGs and varies between 5 % and 15 %. A thorough study of the performance of BSGs with masks in simulations, especially in comparison with unstructured meshes, is yet to come.

## 6. CONCLUSIONS & OUTLOOK

We presented a method for generating block-structured grids which relies on an unstructured triangular mesh as the sole input. The BSGs have a prescribed number of quad blocks refined uniformly into triangular elements. We make use of the Discrete Mesh Optimization (DMO) method for repositioning vertices as we have different quality metrics throughout the method. Masking elements allows representing features much smaller than the block size which is very important when considering complex domain shapes. We evaluate our block-structured grids by comparing them to the unstructured triangular meshes. In future work, we plan to improve the masking process to represent fine details like isthmuses and islands that are even smaller than one element wide. Furthermore, automatic alignment of vertices to interior features will be added.

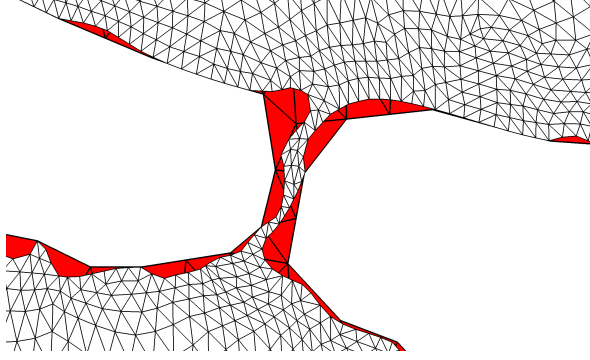
## Acknowledgements

The authors acknowledge financial support by the German Research Foundation (DFG) through grants AI 117/6-1, KO 4641/1-1, and GR 1107/3-1 as well as by the National Centre for High Performance Com-

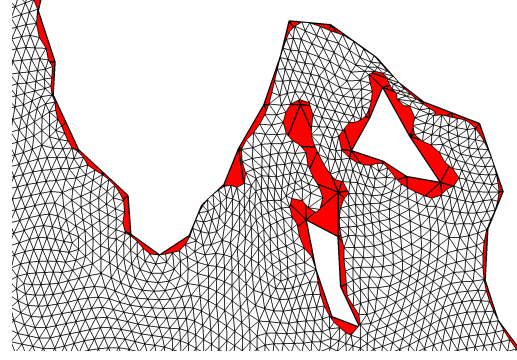
puting (NHR) at the Friedrich-Alexander-University Erlangen-Nuremberg. Furthermore, we acknowledge the work of Jonathan Schmalfuß from the University of Bayreuth who documented and refactored the code for publication.

## References

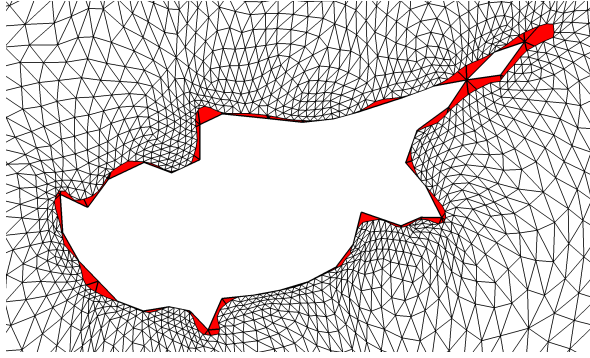
- [1] Gropp W.D., Kaushik D.K., Keyes D.E., Smith B.F. “Performance Modeling and Tuning of an Unstructured Mesh CFD Application.” *SC '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, pp. 34–34. Nov. 2000
- [2] White B.S., McKee S.A., de Supinski B.R., Miller B., Quinlan D., Schulz M. “Improving the computational intensity of unstructured mesh applications.” *Proceedings of the 19th annual international conference on Supercomputing*, ICS '05, pp. 341–350. Association for Computing Machinery, New York, NY, USA, Jun. 2005
- [3] Il'in V. “Integrated Computational Environment for Grid Generation Parallel Technologies.” *Parallel Computational Technologies*, Communications in Computer and Information Science, pp. 58–68. Springer International Publishing, Cham, 2020
- [4] Bank R.E., Smith R.K. “Mesh Smoothing Using A Posteriori Error Estimates.” *SIAM Journal on Numerical Analysis*, vol. 34, no. 3, 979–997, Jun. 1997
- [5] Freitag L., Jones M., Plassmann P. “A Parallel Algorithm for Mesh Smoothing.” *SIAM Jour-*



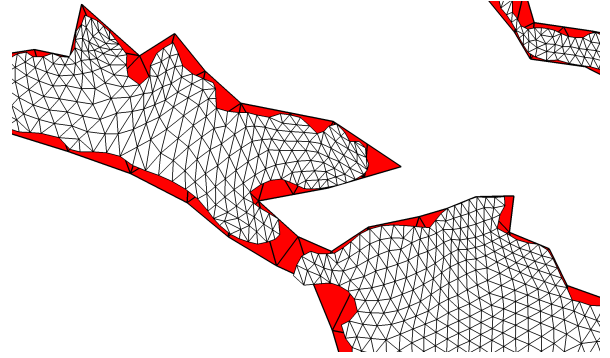
(a) Bosphorus



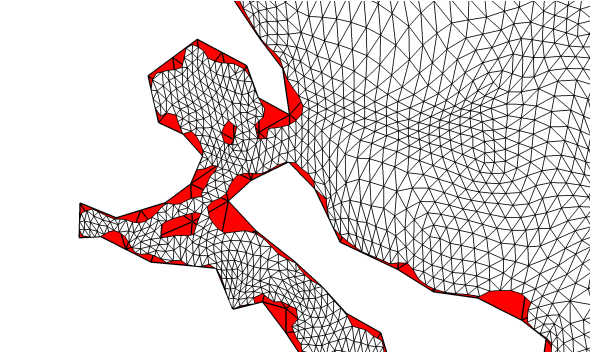
(b) Coast of Croatia



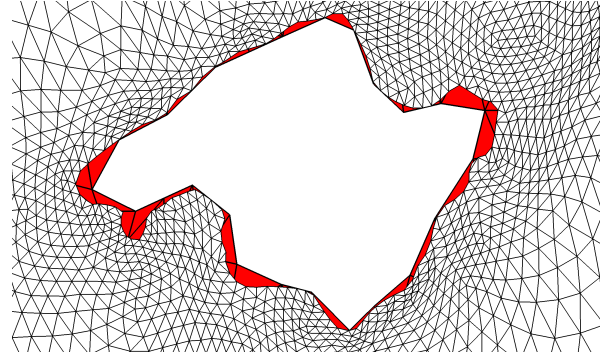
(c) Cyprus



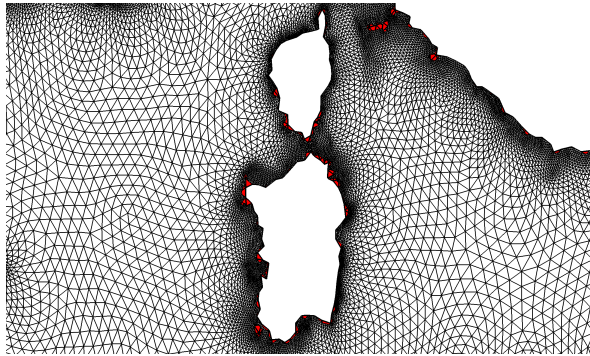
(d) Gulf of Corinth



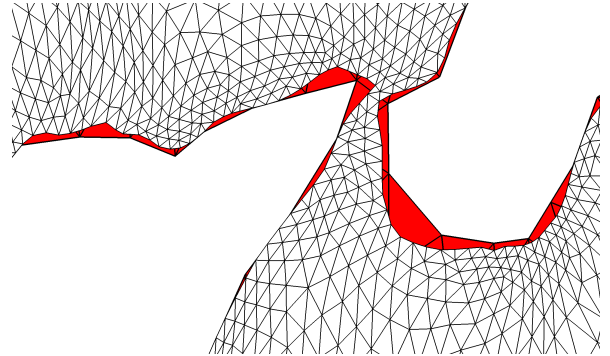
(e) Malian Gulf



(f) Mallorca

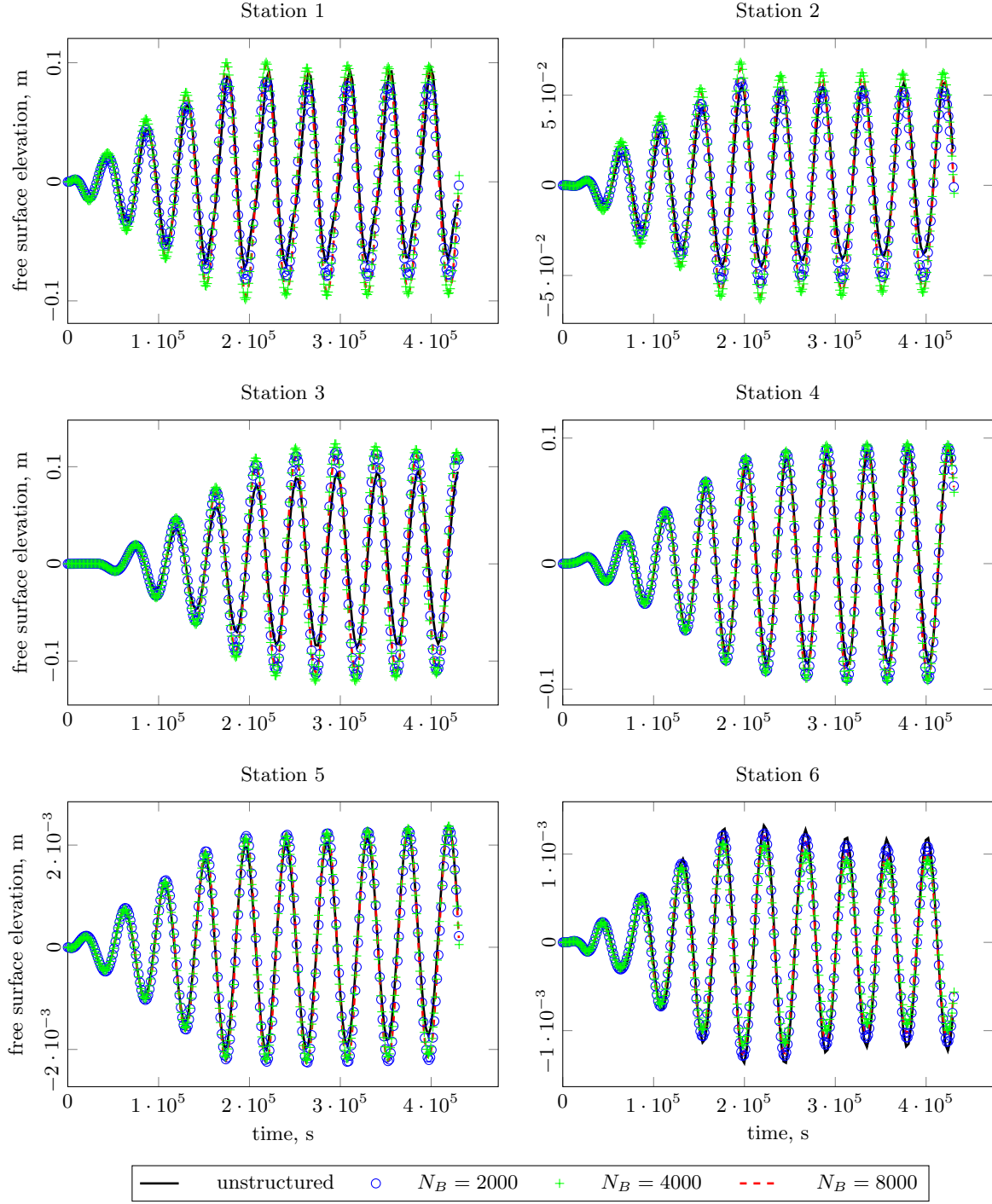


(g) Sardinia



(h) Sicily

**Figure 7:** Detail views of the Mediterranean domain with  $N_B = 8000$ . Red regions are covered by the block grid but are not part of the masked BSG.



**Figure 8:** Free surface elevation at recording stations on Mediterranean.

*nal on Scientific Computing*, vol. 20, no. 6, 2023–2040, Jan. 1999

- [6] Bergen B.K., Hülsemann F. “Hierarchical hybrid grids: data structures and core algorithms for

multigrid.” *Numerical Linear Algebra with Applications*, vol. 11, no. 2-3, 279–291, 2004

- [7] Kohl N., Thönnies D., Drzisga D., Bartuschat D., Rüdte U. “The HyTeG finite-element software

- framework for scalable multigrid solvers.” *International Journal of Parallel, Emergent and Distributed Systems*, vol. 34, no. 5, 477–496, 2019
- [8] Turek S., Göddeke D., Becker C., Buijssen S.H., Wobker H. “FEAST—realization of hardware-oriented numerics for HPC simulations with finite elements.” *Concurrency and Computation: Practice and Experience*, vol. 22, no. 16, 2247–2265, 2010
  - [9] Falgout R.D., Jones J.E., Yang U.M. “The design and implementation of hypre, a library of parallel high performance preconditioners.” *Numerical solution of partial differential equations on parallel computers*, pp. 267–294. Springer, 2006
  - [10] Armstrong C.G., Fogg H.J., Tierney C.M., Robinson T.T. “Common themes in multi-block structured quad/hex mesh generation.” *Procedia Engineering*, vol. 124, 70–82, 2015
  - [11] Fogg H.J., Armstrong C., Robinson T.T. “Multi-Block Decomposition Using Cross-Fields.” *Proceedings of adaptive modelling and simulation, Lisbon*, pp. 254–267, 2013
  - [12] Fogg H.J., Armstrong C.G., Robinson T.T. “Automatic generation of multiblock decompositions of surfaces.” *International Journal for Numerical Methods in Engineering*, vol. 101, no. 13, 965–991, 2015
  - [13] Lim C.W., Yin X., Zhang T., Su Y., Goh C.K., Moreno A., Shahpar S. “Automatic Blocking of Shapes Using Evolutionary Algorithm.” *International Meshing Roundtable*, pp. 169–188. Springer, 2018
  - [14] Lim C.W., Yin X., Zhang T., Selvaraj S.K., Su Y., Goh C.K., Moreno A., Shahpar S. “Towards Automatic Blocking of Shapes using Evolutionary Algorithm.” *Computer-Aided Design*, vol. 120, 102798, Mar. 2020
  - [15] Barrera P., Méndez I. “Parametrization of plane irregular regions: A semi-automatic approach I.” *Numerical Geometry, Grid Generation and Scientific Computing*, pp. 263–279. Springer International Publishing, Cham, 2021
  - [16] Xu G., Li M., Mourrain B., Rabczuk T., Xu J., Bordas S.P.A. “Constructing IGA-suitable planar parameterization from complex CAD boundary by domain partition and global/local optimization.” *Computer Methods in Applied Mechanics and Engineering*, vol. 328, 175–200, Jan. 2018
  - [17] Zint D., Grosso R., Aizinger V., Köstler H. “Generation of Block Structured Grids on Complex Domains for High Performance Simulation.” *Computational Mathematics and Mathematical Physics*, vol. 59, no. 12, 2108–2123, Dec. 2019
  - [18] Boier-Martin I., Rushmeier H., Jin J. “Parameterization of triangle meshes over quadrilateral domains.” *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP ’04, pp. 193–203. Association for Computing Machinery, New York, NY, USA, Jul. 2004
  - [19] Carr N.A., Hoberock J., Crane K., Hart J.C. “Rectangular multi-chart geometry images.” *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP ’06, pp. 181–190. Eurographics Association, Goslar, DEU, Jun. 2006
  - [20] Dong S., Bremer P.T., Garland M., Pascucci V., Hart J.C. “Spectral surface quadrangulation.” *ACM SIGGRAPH 2006 Papers*, SIGGRAPH ’06, pp. 1057–1066. Association for Computing Machinery, New York, NY, USA, Jul. 2006
  - [21] Daniels J., Silva C.T., Cohen E. “Semi-regular Quadrilateral-only Remeshing from Simplified Base Domains.” *Computer Graphics Forum*, vol. 28, no. 5, 1427–1435, 2009
  - [22] Campen M. “Partitioning Surfaces Into Quadrilateral Patches: A Survey.” *Computer Graphics Forum*, vol. 36, no. 8, 567–588, 2017
  - [23] Remacle J.F., Lambrechts J., Seny B., Marchandise E., Johnen A., Geuzainet C. “Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm.” *International Journal for Numerical Methods in Engineering*, vol. 89, no. 9, 1102–1119, 2012
  - [24] Tarini M., Pietroni N., Cignoni P., Panozzo D., Puppo E. “Practical quad mesh simplification.” *Computer Graphics Forum*, vol. 29, no. 2, 407–418, 2010
  - [25] Lengauer C., Apel S., Bolten M., Chiba S., Rüde U., Teich J., Größlinger A., Hannig F., Köstler H., Claus L., Grebhahn A., Groth S., Kronawitter S., Kuckuk S., Rittich H., Schmitt C., Schmitt J. “ExaStencils: Advanced Multigrid Solver Generation.” *Software for Exascale Computing - SPPEXA 2016-2019*, pp. 405–452. Springer International Publishing, Cham, 2020
  - [26] Faghih-Naini S., Kuckuk S., Aizinger V., Zint D., Grosso R., Köstler H. “Quadrature-free discontinuous Galerkin method with code generation features for shallow water equations on automatically generated block-structured meshes.” *Advances in Water Resources*, p. 103552, 2020

- [27] Aizinger V., Dawson C. “A discontinuous Galerkin method for two-dimensional flow and transport in shallow water.” *Advances in Water Resources*, vol. 25, no. 1, 67–84, 2002
- [28] Garland M., Zhou Y. “Quadric-based simplification in any dimension.” *ACM Transactions on Graphics*, vol. 24, no. 2, 209–239, Apr. 2005
- [29] Dey T.K., Edelsbrunner H., Guha S., Nekhayev D.V. “Topology Preserving Edge Contraction.” *Publ. Inst. Math. (Beograd) (N.S.)*, vol. 66, 23–45, 1998
- [30] Garland M., Heckbert P.S. “Surface simplification using quadric error metrics.” *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH ’97, pp. 209–216. ACM Press/Addison-Wesley Publishing Co., USA, Aug. 1997
- [31] Botsch M., Kobbelt L. “A remeshing approach to multiresolution modeling.” *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP ’04, pp. 185–192. Association for Computing Machinery, New York, NY, USA, Jul. 2004
- [32] Zint D., Grosso R. “Discrete Mesh Optimization on GPU.” *27th International Meshing Roundtable*, Lecture Notes in Computational Science and Engineering, pp. 445–460. Springer International Publishing, Cham, 2019
- [33] Bommers D., Lempfer T., Kobbelt L. “Global Structure Optimization of Quadrilateral Meshes.” *Computer Graphics Forum*, vol. 30, no. 2, 375–384, 2011
- [34] Zint D., Grosso R. “A Hybrid Approach to Fast Indirect Quadrilateral Mesh Generation.” *Numerical Geometry, Grid Generation and Scientific Computing*, pp. 281–294. Springer International Publishing, Cham, 2021
- [35] Cialone M.A., Militello A., Brown M.E., Kraus N.C. *COUPLING OF WAVE AND CIRCULATION NUMERICAL MODELS AT GRAYS HARBOR ENTRANCE, WASHINGTON, USA*, pp. 1279–1291. World Scientific Publishing Company, 2003

# LOCAL DECOMPOSITION OF HEXAHEDRAL SINGULAR NODES INTO SINGULAR CURVES

Paul Zhang<sup>1</sup>   Judy (Hsin-Hui) Chiang<sup>2</sup>   Xinyi (Cynthia) Fan<sup>3</sup>   Klara Mundilova<sup>4</sup>

<sup>1</sup>*Massachusetts Institute of Technology, Cambridge, Massachusetts, U.S.A. pzp1@mit.edu*

<sup>2</sup>*University of Illinois at Urbana-Champaign, Urbana-Champaign, Illinois, U.S.A. hsinhui2@illinois.edu*

<sup>3</sup>*Davidson College, Davidson, North Carolina, U.S.A. cyfan@davidson.edu*

<sup>4</sup>*Massachusetts Institute of Technology, Cambridge, Massachusetts, U.S.A. kmundil@mit.edu*

## ABSTRACT

Hexahedral (hex) meshing is a long studied topic in geometry processing with many fascinating and challenging associated problems. Hex meshes vary in complexity from structured to unstructured depending on application or domain of interest. Fully structured meshes require that all interior mesh edges are adjacent to exactly four hexes. Edges not satisfying this criteria are considered singular and indicate an unstructured hex mesh. Singular edges join together into singular curves that either form closed cycles, end on the mesh boundary, or end at a singular node, a complex junction of more than two singular curves. While all hex meshes with singularities are unstructured, those with more complex singular nodes tend to have more distorted elements and smaller scaled Jacobian values. In this work, we study the topology of singular nodes. We show that all eight of the most common singular nodes are decomposable into just singular curves. We further show that all singular nodes, regardless of edge valence, are locally decomposable. Finally we demonstrate these decompositions on hex meshes, thereby decreasing their distortion and converting all singular nodes into singular curves. With this decomposition, the enigmatic complexity of 3D singular nodes becomes effectively 2D.

**Keywords:** hexahedral mesh, singular graph, computational geometry

## 1. INTRODUCTION

Hexahedral meshes are commonly used to model complex geometries and to solve numerical PDEs. The results they produce with tri-linear basis functions are often superior to those produced with linear basis functions on tetrahedral meshes [1, 2]. They can be preferable to other types of meshes for their natural local coordinate systems and have been shown to perform better with quadratic basis functions in the context of nonlinear elasto-plastic simulation [3]. Due to the persistent demand for hex meshes, a variety of methods have been developed to generate them.

Of particular relevance to our work are frame field based methods, where a smooth boundary aligning coordinate system is computed over the domain, followed

by parameterization and hex extraction [4, 5, 6, 7, 8]. Frame field based hex meshing has especially elucidated the significance of singularities within a hex mesh since frame fields inevitably contain singular structures that are reflected in the resulting mesh. The singularities of a frame field and hex mesh typically consist of a set of singular curves that join up in space at singular nodes. These nodes and curves form the singular graph of a field or mesh as illustrated in Figure 1. Since hex meshable singularities are a subset of frame field singularities, much attention has been devoted to the restriction and correction of singularities in frame field computation [9, 10]. Other works have derived conditions and algorithms to compute frame fields obeying singular constraints [11, 12].

Various works have also targeted the enumeration or



simplification of hex mesh singularities. [13, 14, 15] derive algorithms to simplify the singular structures of hex meshes with collapse operations on a coarsened mesh. [11] provide an enumeration algorithm for all hex mesh singular nodes, as well as an exhaustive list of the most practically relevant singular node types. While this list only contains eight singular nodes, they already form complex junctions that are challenging to parse or manipulate.

In contrast to singular nodes, singular curves are simpler to understand, since their local structure is only a 2D singularity extruded into 3D. In the fully 2D setting, cross field and quadrilateral (quad) mesh singularities are significantly easier to visualize, and in the case of parameterized cross fields, singularities are governed completely by a few simple conditions [16]. In quad meshes, singular vertex pairs are shown to move almost fluidly within a quad mesh [17]. None of these results obviously translates to hex meshes, where 3D singular nodes exist as junctions of multiple 2D singularities.

In this paper, we investigate the structure of hex mesh singular nodes. We uncover that singular nodes can be simplified by pulling their constituent elements apart into singular curves. Our results show theoretically and empirically that singular nodes can be removed from a hex mesh thus reducing the complexity of any local neighborhood in a hex mesh. Our contributions are as follows:

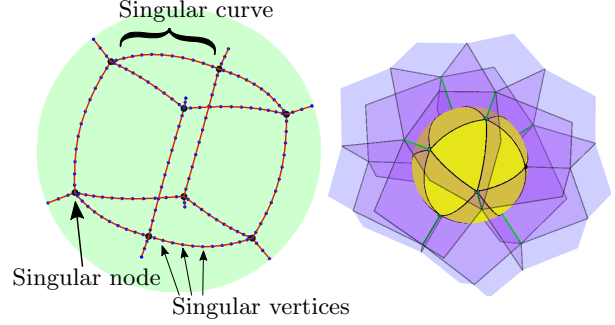
- We show by construction that all eight of the most practically relevant singular nodes are decomposable into just singular curves.
- We show that all singular nodes, regardless of valence, are locally decomposable.
- We apply our decompositions to hex meshes demonstrating that entire singular graphs can be separated into independent singular curves.

## 2. PRELIMINARIES

Our work is motivated by the following question. What if a singular node is formed when singular curves just barely skim past each other? If that were the case, then we could separate the curves with a sheet and increase its thickness to force the curves away from each other, thereby untangling the singular node. To formalize this idea, we begin with the following definitions.

### 2.1 Singular Vertices, Curves, and Nodes

We denote a hex mesh as  $\{\mathcal{V}, \mathcal{H}\}$  where  $\mathcal{V}$  is a list of vertices embedding the mesh and  $\mathcal{H}$  is a list of hexes of

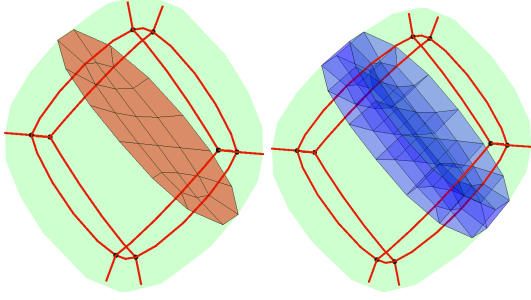


**Figure 1:** (Left) The singular graph of a hex mesh of a sphere is shown. Singular edges are colored red, singular nodes are large black circles and singular vertices are small blue vertices. (Right) A close-up view of a singular node. Faces adjacent to the singular node are displayed in purple. A yellow sphere is overlaid on top of the singular node. Its intersection with the local hex mesh partitions the sphere into triangular regions.

the mesh. Let  $\mathcal{F}$  denote the quadrilateral faces of the mesh,  $\mathcal{E}$  denote the edges of the mesh, and  $\deg(e \in \mathcal{E})$  denote the number of hexes adjacent to edge  $e$ , i.e., its degree or valence. A *singular edge* is an interior edge  $e$  satisfying  $\deg(e) \neq 4$ . We will not treat cases where  $\deg(e) \leq 2$  since these are not typically accepted as valid hex meshes. We will also not consider singular boundary edges in this paper but refer the interested reader to [11] for the corresponding definition. For our purposes, one can ignore boundary singularities or push them all to the interior by adding one layer of padding to the hex mesh boundary. A *singular vertex* is a vertex of the mesh that is adjacent to any singular edge. A *singular node* is a vertex of the mesh that is adjacent to more than two singular edges. A *singular curve* is an alternating sequence of singular edges and singular vertices that either forms closed cycles or, ends at a singular node or boundary vertex. Note that we have chosen to deviate from the language of [11] by distinguishing singular nodes from singular vertices. Singular nodes are reserved for the junctions of multiple singular curves and will be the primary focus of this work. Figure 1 depicts a summary of this terminology.

For a singular node  $v \in \mathcal{V}$ , we denote  $\mathcal{T}(v)$  as the triangle mesh in bijection with that node according to [6]. This bijection is formed by intersecting the singular node of the hex mesh with an infinitesimally small sphere. Since the intersection of a corner of a hex with a sphere forms a triangle, the hexes adjacent to the singular node partition the sphere into triangular regions thus forming a sphere triangulation. This is depicted in Figure 1. The sphere triangulation encodes the *singular node type*. If two sphere triangulations are isomorphic, then their singular nodes are of the same type. The *signature* of a singular node is a





**Figure 2:** (Left) Red quads indicate a sheet inside of a hex mesh of an ellipsoid. Red curves depict its singular graph. The sheet is manifold with boundary on the boundary of the hex mesh. (Right) Blue quads indicate all faces of the inflated sheet.

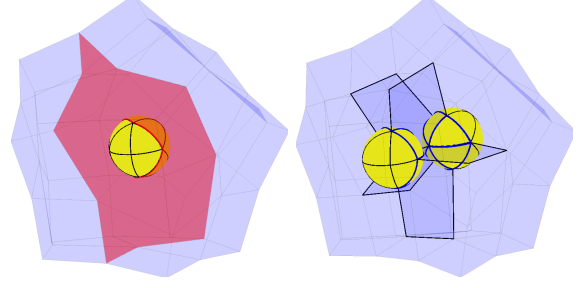
list of numbers indicating how many adjacent singular edges are of each degree. Since singular edges have degree 3 or higher, the signature of a node starts with the number of edges with degree 3. For singular nodes whose adjacent singular edges have only valence 3, 4, or 5, the signature also uniquely encodes the singular node type [11]. For this reason we will frequently identify singular nodes by their signature e.g.  $(4,0,0)$  is the signature identifying the singular node type generated by subdividing a single tetrahedron into four hexes as illustrated in Figure 4

## 2.2 Sheet Inflation

Let a *sheet*  $Q \subset \mathcal{F}$  be a manifold quad mesh whose boundary is a subset of the boundary of  $\mathcal{H}$ . A *sheet inflation* based on sheet  $Q$  is a mesh modifying operation by which each  $q \in Q$  is thickened from a quad face to a hex cell [18]. A sheet inflation operation is depicted in Figure 2

In the case that the inflated sheet passes through a singular node, the singular type of that node can change. One can interpret this as the sheet passing an infinitesimally small gap between singular curves and forcing them apart, or as cutting a singular node into separate pieces. As singular nodes are often more easily visualized as sphere triangulations we describe here how sheet inflation through a singular node corresponds to a *splitting* of the sphere triangulation. This splitting is illustrated by Figure 3

Given a singular node  $v$ , and a sheet  $Q \subset \mathcal{F}$  that passes through  $v$ , the faces of  $Q$  map to edges of  $\mathcal{T}(v)$ . These edges trace out a cycle in the graph of  $\mathcal{T}(v)$  effectively partitioning the sphere into two disk triangulations  $\mathcal{D}_1$  and  $\mathcal{D}_2$ . When the sheet is thickened into a layer of hexes, these two disk triangulations are cut apart. Then the disks are patched into sphere triangulations again by adding one new vertex to each disk and attaching triangles from the boundary of each disk



**Figure 3:** (Left) The yellow sphere triangulation indicates the structure of a singular node. Red quads indicate a sheet intersecting the singular node. The sheet intersects the sphere triangulation on a cycle of red curves that divide the sphere triangulation into two disks. (Right) Two singular nodes are visualized from inflation of the red sheet on the left. Sphere triangulations of the resulting two nodes are shown. Blue quad faces indicate newly created faces from the inflation. Blue edges indicate newly created edges in each sphere triangulation.

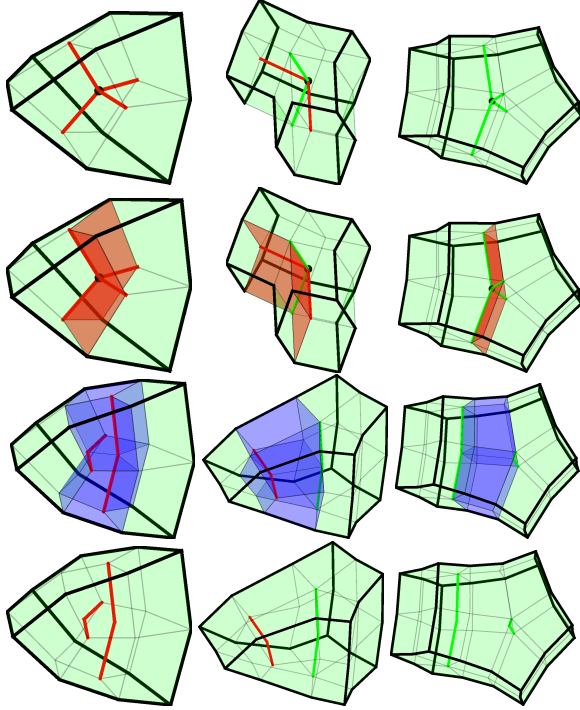
to their respective new vertex. This splitting operation is illustrated in Figure 3. The end result is two sphere triangulations one built from  $\mathcal{D}_1$  and one built from  $\mathcal{D}_2$ .

## 3. SINGULAR NODE DECOMPOSITION (VALENCE 3,4,5)

We are now equipped with the mechanism by which all practically relevant singular nodes can be decomposed into singular curves. These nodes are enumerated in [11] Figure 6] and will also be shown at the beginning of each respective decomposition. Various singular decompositions will be depicted throughout the remainder of this paper. Valence 3 singular curves will be red, valence 5 singular curves will be green and higher valence singular curves will be blue. We omit drawing interior regular edges to minimize clutter. Since these operations can be challenging to understand from static images, we also include supplemental videos for many of the decompositions.

In the first column of Figure 4 we start with the  $(4,0,0)$  singular node, which consists of four valence 3 singular curves joined at a junction. With a single sheet inflation, this singular node is revealed to actually be two valence 3 singular curves that pass each other orthogonally. A similar theme follows for  $(2,2,2)$  in the second column which is revealed to be a valence 3 and a valence 5 singular curve passing each other orthogonally. Finally in the third column, the  $(0,4,4)$  node decomposes into two valence 5 singular curves passing each other orthogonally.

From these three examples, it is tempting to think all singular nodes may consist of singular curves glued to-



**Figure 4:** From left to right the  $(4,0,0)$ ,  $(2,2,2)$ , and  $(0,4,4)$  singular nodes are depicted. Top to bottom indicates steps to decompose each singular node. Red quads indicate the sheet to be inflated. Blue quads indicate faces of the newly inflated hexes. Red(Green) edges are valence 3(5) singularities. One sheet inflation is sufficient to decompose each of these nodes.

gether orthogonally. One might conclude as well that the number of singular curves of a particular valence meeting at a singular node from this construction must be even. The  $(1,3,3)$  singular node, shown in [Figure 5](#) presents a curious counterexample. Since it consists of one valence 3 singular curve and three valence 5 singular curves, it is impossible to decompose these into two singular curves passing each other orthogonally in a valid hex mesh.

This conundrum is resolved by realizing that one of the regular edges adjacent to this singular node is actually a pair of valence 3 and valence 5 singular curves, glued together in parallel. Another way to understand this node is that one valence 5 singular curve has split an otherwise parallel pair of valence 3 and 5 curves. The decomposition of this node into one valence 3 and two valence 5 singular curves is illustrated in [Figure 5](#).

In [Figure 6](#) we decompose the  $(0,3,6)$  singular node into three valence 5 singular curves. It is also valuable to think of the  $(0,3,6)$  node as a combination of two  $(0,4,4)$  singular nodes. In this way, one only needs to decompose a singular node into constituent nodes that are already known to be decomposable. The fourth image of [Figure 6](#) shows exactly this decomposition which we will notate as:

$$(0, 3, 6) = (0, 4, 4) +_5 (0, 4, 4)$$

The subscript 5 on the plus symbol denotes that two singular nodes are joined along a valence 5 edge, followed by an inverse sheet inflation (sheet collapse). This notation serves only as a shorthand and does not uniquely encode how to glue two singular nodes together. It serves more as a recipe than an equation with any algebraic properties. For completion, we show the rest of the decomposition of the constituent  $(0,4,4)$  nodes.

The  $(0,2,8)$  singular node is decomposed in [Figure 7](#) into four valence 5 curves. In the fourth image, we see that

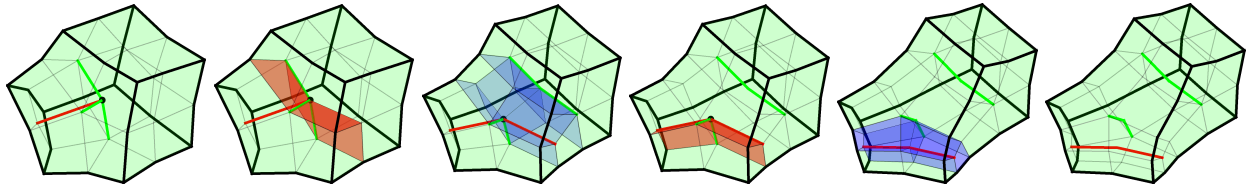
$$(0, 2, 8) = (0, 3, 6) +_5 (0, 4, 4)$$

and both constituent singular nodes have already been shown to be decomposable. For completion, the rest of the decomposition steps are also shown.

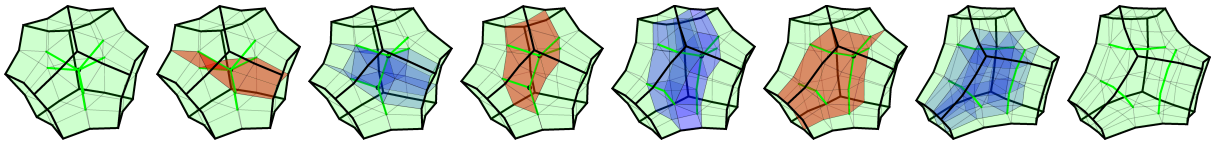
The  $(2,0,6)$  singular node is decomposed in [Figure 8](#) into four valence 5 and two valence 3 curves. By the fourth image we see

$$(2, 0, 6) = (1, 3, 3) +_4 (1, 3, 3).$$

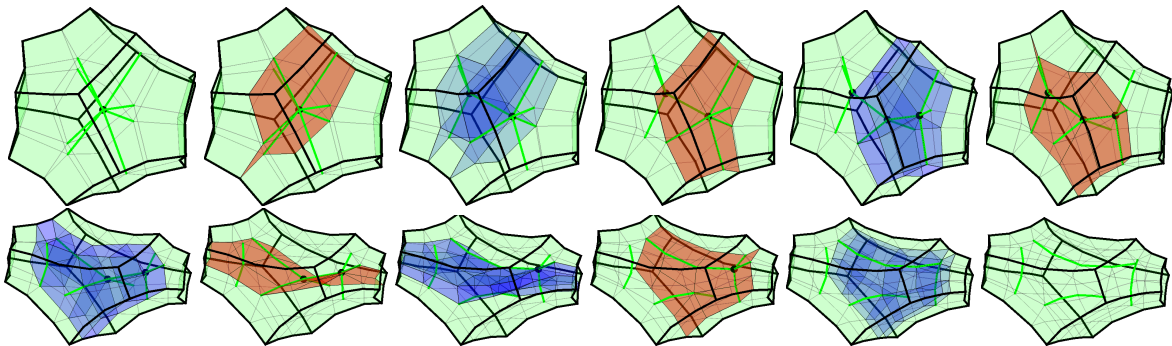
The rest of this decomposition is still interesting however as it introduces a valence 6 singularity in the fifth image and a singular node with signature  $(1,3,3,1)$ . This valence 6 singular curve is removed in image 9 by decomposing it into two valence 5 curves.



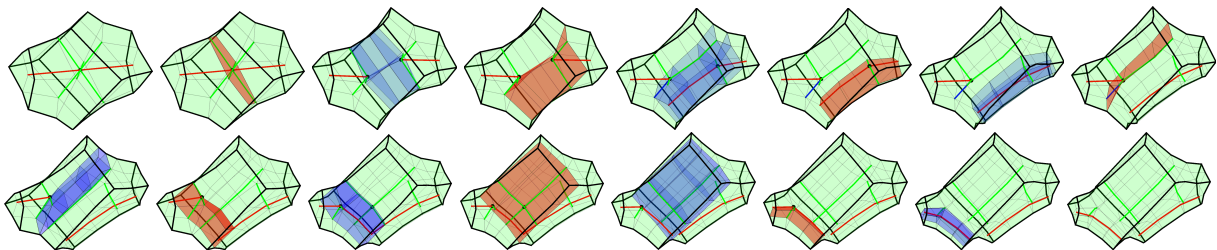
**Figure 5:** The  $(1,3,3)$  singular node is decomposed into two valence 5 and one valence 3 curve via two sheet inflations.



**Figure 6:** The  $(0,3,6)$  singular node is decomposed into three valence 5 curves via three sheet inflations.



**Figure 7:** The  $(0,2,8)$  singular node is decomposed into four valence 5 curves via five sheet inflations.



**Figure 8:** The  $(2,0,6)$  singular node is decomposed into four valence 5 and two valence 3 curves via seven sheet inflations.

Finally the  $(0,0,12)$  singular node decomposition is shown in [Figure 9](#) to become six valence 5 curves. This singular node is especially interesting as we were unable to show a decomposition of the form

$$(0, 0, 12) = (0, 4, 4) +_n (0, 2, 8).$$

Even though the number of singular curves present is sufficient, we were not able to perform an inverse sheet inflation between  $(0,4,4)$  and  $(0,2,8)$  to obtain  $(0,0,12)$ . This shows that the order in which singular curves are combined matters. As this figure is especially complex to comprehend, we offer the following roadmap of how the decomposition is performed.

$$(0, 0, 12) = \underbrace{((0, 3, 6) +_5 (0, 3, 6))}_{(0,2,8,1)} +_6 \underbrace{((0, 4, 4) +_5 (0, 4, 4))}_{(0,4,4,1)}$$

#### 4. DECOMPOSING GENERAL SINGULAR NODES

Given that the eight singular nodes of valence 3, 4, or 5 are decomposable into singular curves, a natural next question is whether decomposition extends to higher valence singular nodes. In fact, the decomposition of the  $(0,0,12)$  and  $(2,0,6)$  both already required decomposing singular nodes with valence 6:  $(0,4,4,1)$ ,  $(0,2,8,1)$  and  $(1,3,3,1)$ . We will refer to previously known decomposable singular nodes and their associated sphere triangulations as *base cases*. To generalize decomposability of singular nodes we offer the following result.

**Proposition 1.** Given a sphere triangulation  $\mathcal{T}$  with some vertex  $u$  of degree larger than 5, there exists a splitting such that either the number of vertices in both resulting triangulations decreases or the resulting triangulations are base cases.

*Proof.* The local neighborhood of  $u$  is an umbrella  $\mathcal{U}$  of at least 6 triangles. The boundary of this umbrella is a cycle of at least 6 vertices denoted by  $\mathcal{C}$ . To construct a splitting of  $\mathcal{T}$  into triangulations of fewer vertices, we need a pair of vertices  $a$  and  $b$  adjacent to  $u$  that are at least 3 edges apart from each other in  $\mathcal{C}$  such that there is path  $p$  from  $a$  to  $b$  through the interior of  $\mathcal{T} - \mathcal{U}$ . This construction is illustrated in [Figure 10](#). The sequence of edges  $[(ua), p, (bu)]$  partitions  $\mathcal{T}$  into  $\mathcal{D}_1$  and  $\mathcal{D}_2$  where each disk triangulation has at least 2 interior vertices. Since splitting a sphere triangulation replaces all vertices on the interior of either side with just one new vertex each, both resulting triangulations will have fewer vertices than  $\mathcal{T}$ . For readability, we leave more detailed construction of the splitting to supplementary materials.  $\square$

**Algorithm 1** Decomposes all singular nodes of a hex mesh into singular curves.

---

```

1: procedure DECOMPOSE-SINGULAR-GRAPH( $\mathcal{H}$ )
2:   do
3:      $N \leftarrow \text{GETRANDOMSINGULARNODE}(\mathcal{H})$ 
4:     if ONLYHASVALENCE345( $\mathcal{H}, N$ ) then
5:        $C \leftarrow \text{GETHARDCODEDCUT}(\mathcal{H}, N)$ 
6:     else
7:        $C \leftarrow \text{GETGENERALCUT}(\mathcal{H}, N)$ 
8:     end if
9:      $S \leftarrow \text{PROPAGATECUT}(\mathcal{H}, C)$ 
10:     $H \leftarrow \text{SHEETINFLATION}(\mathcal{H}, S)$ 
11:  while  $N \neq \emptyset$ 
12:  return  $\mathcal{H}$ 
13: end procedure

```

---

Applying the splitting in [Prop. 1](#) could result directly in base cases, where the rest of the decomposition is already known. If the splitting does not result in base cases, then it produces triangulations with fewer vertices. This can be repeated until there are not enough vertices to have a degree 6 vertex. Since sheet inflation at a singular node corresponds to splitting of a sphere triangulation, [Prop. 1](#) allows us to find a sequence of sheets whose inflation results in singular nodes that have lower than valence 6 singular edges. We have already enumerated singular decompositions for all singular nodes with valence lower than 6 and can therefore decompose any singular node into singular curves.

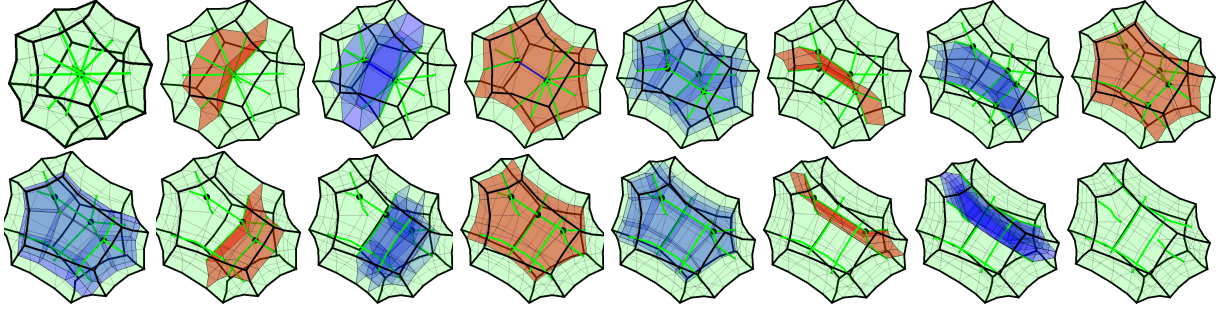
A limitation of [Prop. 1](#) is that it restricts attention to individual singular nodes while ignoring the full singular graph of the mesh. It can be challenging to extend a sheet known locally around a singular node to the rest of the hex mesh while guaranteeing no self-intersection occurs. We present our simplistic solution to extending sheets in [subsection 5.1](#) and leave more careful consideration of how to avoid self-intersection to future work.

## 5. RESULTS

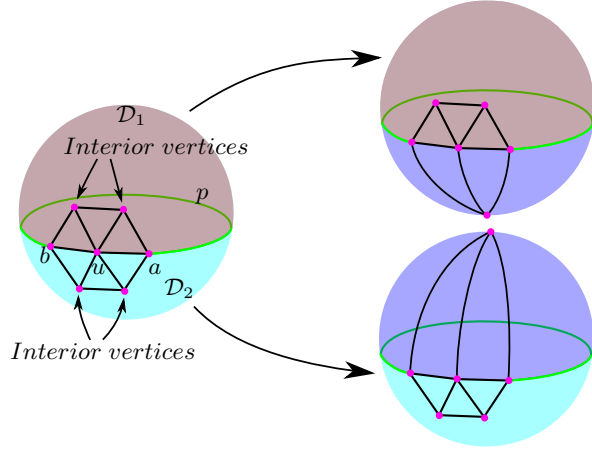
### 5.1 Singular Graph Decomposition

We develop a procedure to perform singular mesh decompositions on general hex meshes. Pseudocode for this procedure is given in [Alg. 1](#) and [Alg. 2](#). First, we randomly select a singular node. For any singular node with valence restricted to 3, 4, or 5, we hard code a subset of faces adjacent to the node to be inflated. If the node has valence 6 or higher, we use [Prop. 1](#) (denoted `GetGeneralCut` in [Alg. 1](#)) to select these faces. These faces form a partial sheet that decomposes the initially selected node, but need to be extended through the rest of the mesh in order to be inflatable.





**Figure 9:** The  $(0,0,12)$  singular node is decomposed into six valence 5 curves via seven sheet inflations.



**Figure 10:** Illustration of how to find a cycle such that splitting along that cycle results in two sphere triangulations, each with fewer vertices. The only requirement is that there is a vertex  $u$  of degree  $\geq 6$ . The required cycle is then  $[(ua), p, (bu)]$ .

Next we propagate the partial sheet throughout the hex mesh following [Alg. 2](#). Let a face be *parallel* to the partial sheet if they share a regular edge but share no adjacent hexes. We greedily add parallel faces to the partial sheet until no more parallel faces can be found. Next we look for any interior singular vertices on the boundary of the partial sheet. If such a vertex is found, then we compute the smallest number of new faces that need to be added to the partial sheet so that its boundary excludes this singular vertex. This is denoted by Put-v-In-S in [Alg. 2](#) and is equivalent to a graph shortest path computation on the triangulation representing this singular vertex.

These two steps are repeated until no more parallel faces can be found, and the boundary of the partial sheet is entirely on the boundary of the hex mesh. If at any stage of the algorithm, the partial sheet became non-manifold then the sheet propagation algorithm has failed. If the sheet is manifold then we in-

**Algorithm 2** Propagates a partial sheet into a full sheet recursively.

---

```

1: procedure PROPAGATECUT( $\mathcal{H}, S$ )
2:    $\mathcal{Q} \leftarrow \text{GETFACES}(\mathcal{H})$ 
3:   while  $\exists f \in \mathcal{Q} : \text{PARALLEL}(\mathcal{H}, S, f)$  do
4:      $S \leftarrow S \cup f$ 
5:   end while
6:    $\mathcal{V} \leftarrow \text{GETVERTICES}(\mathcal{H})$ 
7:    $\mathcal{V}_S \leftarrow \text{GETINTERIORSINGULARVERTICES}(\mathcal{H})$ 
8:   if  $\exists v \in (\partial S \cap \mathcal{V}_S)$  then
9:      $S \leftarrow \text{PUT-V-IN-S}(\mathcal{H}, S, v)$ 
10:  else
11:    if  $\text{NONMANIFOLD}(\mathcal{H}, S)$  then
12:      return ERROR
13:    else
14:      return  $S$ 
15:    end if
16:  end if
17:  return PROPAGATECUT( $\mathcal{H}, S$ )
18: end procedure

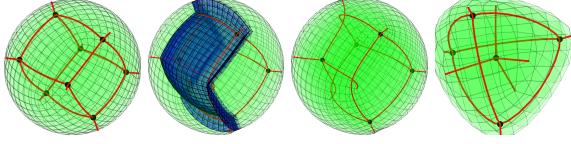
```

---

flate it resulting in the decomposition of at least one singular node. All results shown were generated by [Alg. 1](#)

Applying our decomposition to a hex mesh of a sphere reveals that it has the same singular graph structure as that of a padded tetrahedron. [Figure 11](#) shows this correspondence where inflating one sheet that passes through seven singular nodes, simultaneously decomposes three of them. The end result is a singular graph composed of four  $(4,0,0)$  singular nodes. One of these nodes has singular curves that all connect directly to the boundary. The other four of these nodes connect to each other and the boundary via valence 3 singular curves in a tetrahedral arrangement. This singular graph is exactly what one obtains by padding a hex mesh of a regular tetrahedron i.e. padding a  $(4,0,0)$  node.

Changing how the sheet cuts through the singular graph produces different intermediate and final singu-



**Figure 11:** (Left) Hex mesh of sphere with singular graph. (Mid-left) Blue hexes are newly inflated hexes. (Mid-right) Hex mesh post-inflation. (Right) Singular graph of a padded hex mesh of a tetrahedron. The last two images have topologically equivalent singular graphs.

lar graphs. In [Figure 12](#) we decompose a padded cube in two different sequences and show the their intermediate singular graphs. To improve clarity, we provide schematics of a subset of the singular graphs. The ending singular graphs from both sequences are also topologically distinct i.e. no purely geometric deformation maps one singular graph into the other. They do however appear to invariably contain a single singular cycle.

The first sheet inflation of the second sequence results in the same singular graph as a padded hex mesh of a triangular prism: a padded  $(2,3,0)$ . Since the hex mesh of a sphere has the same singular graph as the padded cube, these results indicate that singular graphs for a padded cube, padded tet, and padded triangular prism are identical up to a series of sheet inflation and collapses.

In [Figure 13](#) we apply our decomposition to more complex singular graphs. The first two rows depict the decomposition of the **G1** hex mesh. The starting singular graph consists of 12 nodes connected by 36 singular curves. This graph is successfully decomposed into seven singular curves, one of which is a closed cycle. The last two rows depict the decomposition of the **G2** hex mesh. The starting singular graph consists of 16 nodes connected by 40 singular curves. This graph is successfully decomposed into 12 singular curves, two of which are closed cycles. While the starting singular graphs are different, both meshes are fully decomposed with the same number of sheet inflations.

Finally, we apply our decomposition in [Figure 14](#) to the cactus mesh from [\[19\]](#). While the mesh starts with only singularities of valence 3, 4 and 5, the decomposition results in intermediate singular graphs with nodes of signature  $(2,3,0,2)$ . The final configuration is seen to contain singular curves of valence 6. Since our goal is only to remove singular nodes, we terminate with valence 6 curves.

## 5.2 Scaled Jacobians

The minimum scaled Jacobian of a hex mesh is a common metric by which to evaluate distortion of the mesh

[\[20\]](#). We maximize the minimum scaled Jacobian before and after singular decomposition of each singular node with free boundaries and present the resulting minimum scaled Jacobians in [Table 1](#). Unsurprisingly, singular nodes have lower scaled Jacobians than singular curves.

By symmetry, the minimum scaled Jacobian of any hex mesh, regardless of resolution, containing a  $(4,0,0)$  node is upper bounded by  $\frac{4}{3\sqrt{3}} = .7698$ . The same bound for a hex mesh containing a  $(0,0,12)$  node is  $\frac{\sqrt{2(5+\sqrt{5})}}{5} = .761$ . These bounds are exactly attained in [Table 1](#) for the  $(4,0,0)$  and  $(0,0,12)$  nodes. The same upper bound computed for meshes containing valence 3 singular curves is  $\sin(\frac{2\pi}{3}) = .866$  and for meshes containing valence 5 singular curves is  $\sin(\frac{2\pi}{5}) = .951$ .

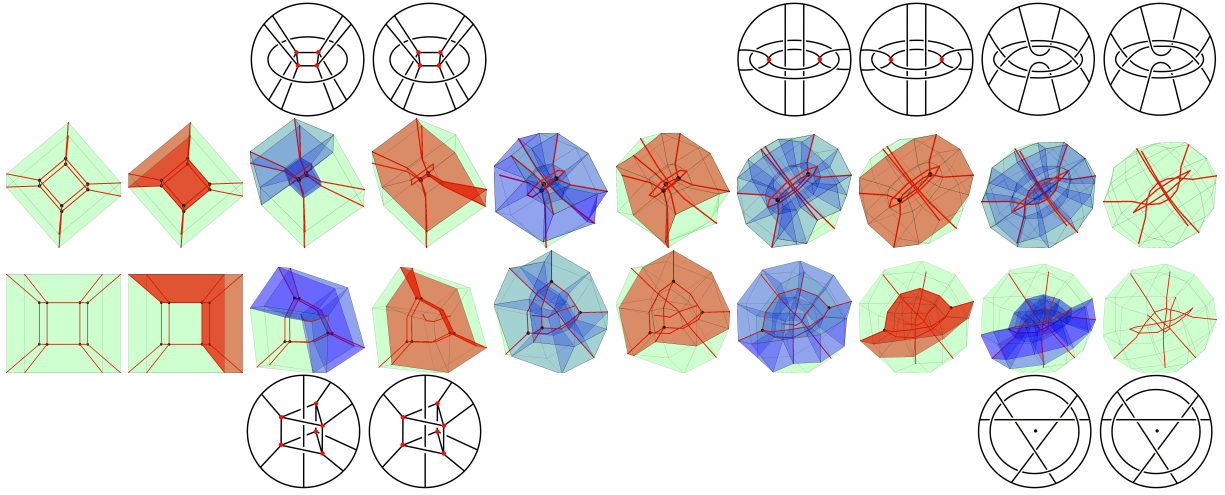
By performing a sheet inflation to split singular nodes into singular curves, the minimum scaled Jacobian of the  $(4,0,0)$  node is increased to .86, almost the theoretic upper bound. For  $(0,4,4)$  as well, decomposing the singular node into two valence 5 curves brings the minimum scaled Jacobian to almost the theoretic upper bound. Decomposing  $(0,0,12)$  node into six valence 5 curves brings significant improvement to the minimum scaled Jacobian, though it is not as close to the theoretic upper bound due to interactions between singular curves.

Moving towards full singular graphs, we perform the same scaled Jacobian optimization for a sphere mesh. Maximization of its minimum scaled Jacobian results in a value of .768, close to the upper bound for any mesh containing a  $(4,0,0)$  node. We apply our decomposition to this mesh and re-optimize its scaled Jacobian resulting in a significant improvement to .849. We run the same optimizations on the padded tetrahedron, **G1**, and **G2** resulting in similar increases in the minimum scaled Jacobian. These results are summarized in [Table 1](#).

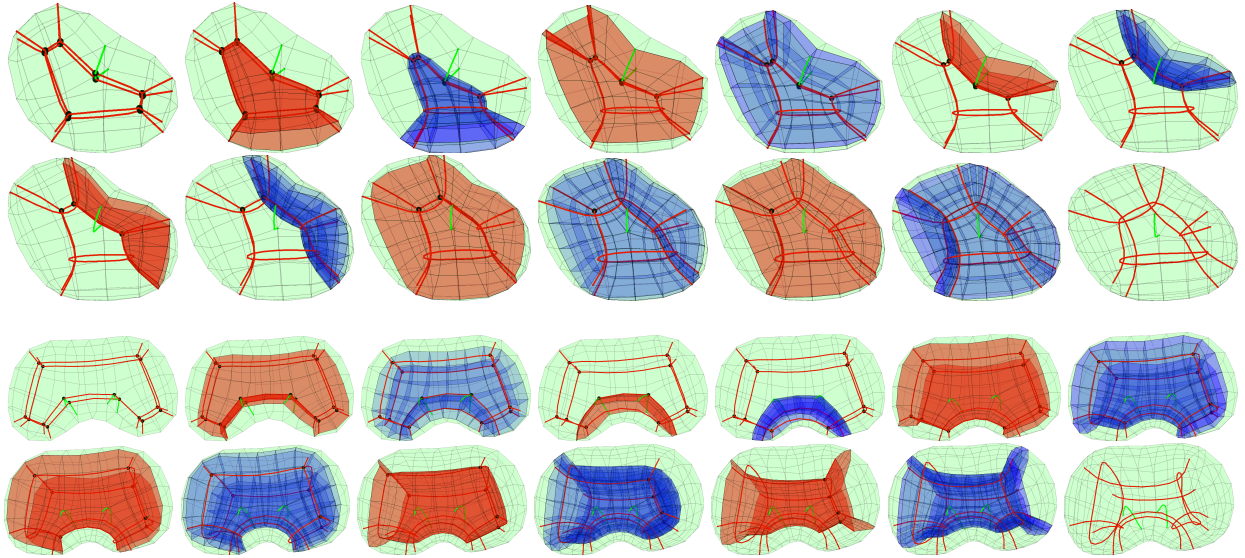
## 6. CONCLUSIONS AND FUTURE WORK

This paper presents singular nodes as the result of gluing singular curves together at a point and shows that the reverse can be done via sheet inflation to untangle singular nodes into simple singular curves. This removes the 3D complexity of singular nodes leaving meshes with lower distortion. We demonstrate this procedure on a variety of meshes showing in all cases that no singular nodes are left behind.

The main limitation of our work is that the local sheets we prescribe for decomposing a singular node are not guaranteed to propagate globally while avoiding self-intersection. This can result in the inability to decompose a singular graph by removing all of its singular

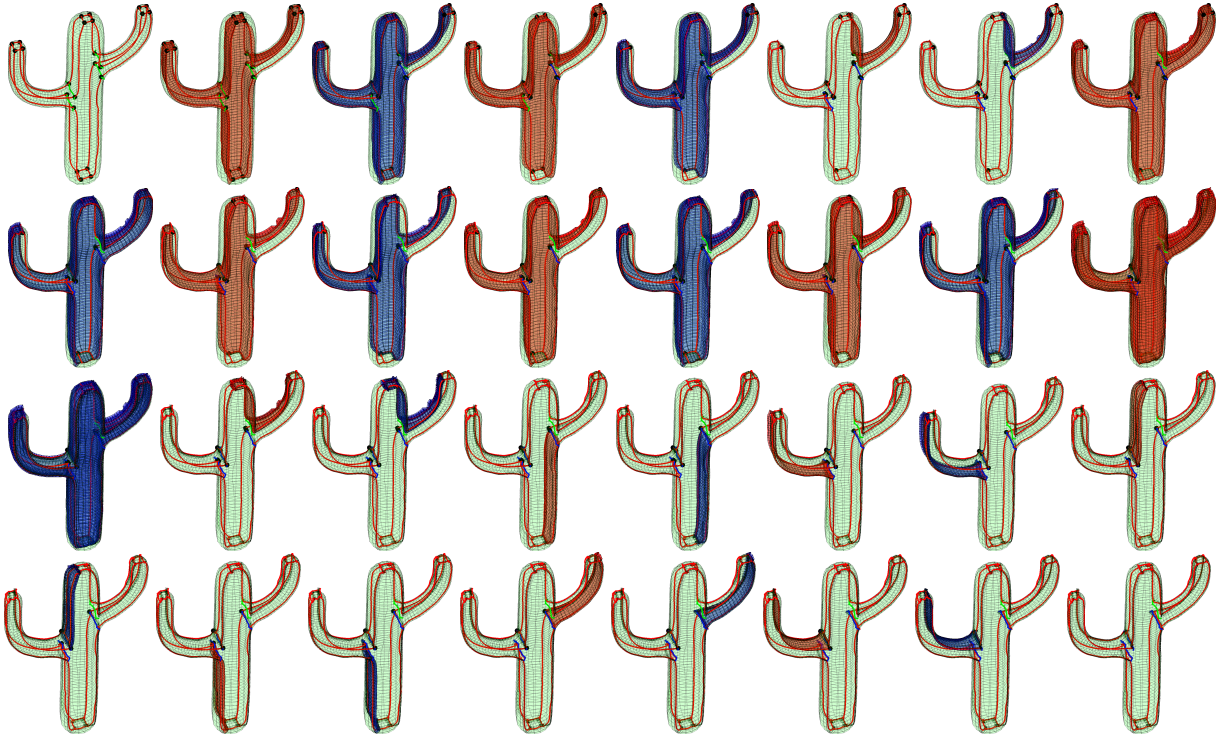


**Figure 12:** We show two sequences of singular graph decomposition starting from the same hex mesh on the left to a fully decomposed singular graph on the right. The first row indicates select singular graph schematics for the first sequence. The last row indicates select singular graph schematics for the second sequence. Even though both singular graphs start out identical, the ending singular graphs are different due to different sheet inflations.



**Figure 13:** We apply singular decomposition to the **G1** and **G2** hex meshes. The first two rows correspond to the sequence of singular graphs from decomposing **G1**. The last two rows correspond to the sequence of singular graphs from decomposing **G2**. The number of singular nodes decreases each sheet inflation ultimately resulting in a singular graph with no nodes at all.





**Figure 14:** We apply singular decomposition to the cactus mesh from [19]. The number of singular nodes decreases with each sheet inflation ultimately resulting in a singular graph with no nodes at all. While the original singular graph consisted of only valence 3, 4 and 5 nodes, intermediate singular graphs from this sequence contain singular nodes with signature  $(2,3,0,2)$ . The fully decomposed singular graph has valence 6 curves.

Mesh	Original	Decomposed	UpperBound
(4,0,0)	0.769	0.86	0.866
(2,2,2)	0.807	0.862	0.866
(0,4,4)	0.896	0.943	0.951
(1,3,3)	0.822	0.865	0.866
(0,3,6)	0.863	0.939	0.951
(0,2,8)	0.82	0.937	0.951
(2,0,6)	0.745	0.856	0.866
(0,0,12)	0.761	0.926	0.951
Sphere	0.768	0.849	0.866
Padded Tet	0.715	0.812	0.866
<b>G1</b>	0.769	0.811	0.866
<b>G2</b>	0.757	0.820	0.866
Ellipsoid	0.767	0.825	0.866

**Table 1:** For various hex meshes, we indicate the maximized minimum scaled Jacobian before and after singular decomposition. The first column indicates the mesh, the second column indicates before singular decomposition, and the third column indicates after. The fourth column indicates a theoretic upper bound on the minimum scaled Jacobian for the decomposed mesh. It essentially indicates the presence of a valence 3 or 5 singular curve. The maximized minimum scaled Jacobian is invariably higher post singular decomposition.

nodes. We expect that a valid sheet inflation can always be found and leave its efficient computation to future work.

While our method decreases the number of singular nodes in a mesh, its *base complex* [14] may increase in size. This tradeoff should be considered by the user as they may have to choose between a larger scaled Jacobian or maintaining a small number of base complex cells.

Our results can be extended to design new ways of modifying the singular graph of a mesh. Instead of only decomposing nodes into curves, one can *rewire* singular curves by merging them at a node with sheet collapse, and decomposing them in a different way from how they were combined. For example, consider the (4,0,0) node in Figure 4. Its sphere triangulation is a tetrahedron which contains three distinct cycles of length four. Therefore, it is possible to bring two valence 3 singular curves together to form a (4,0,0) node and split them apart again in three distinct ways. Each one results in a different singular graph, none of which require introducing new singularities.

Many works aim to build minimal degree smooth parameterizations of quad meshes with singularities [21, 22]. These methods do not clearly generalize to the volumetric case where singular nodes may suffer decreased continuity from methods designed for 2D singularities. A promising approach following our work is then to decompose any given singular graph

so that no singular nodes exist. We expect that it is easier to adapt quad mesh singular parameterization methods to singular curves that are just 2D singularities extruded into 3D than it is to adapt parameterization methods for singular nodes. Even if one derived a singular node parameterization method for a specific singular node type, there is no guarantee that it extends to any other node type. This problem is made easier by only needing to consider singular curves after decomposition.

## ACKNOWLEDGEMENTS

This project was launched at the Summer Geometry Initiative (SGI) 2021, supported by National Science Foundation grant DMS-2103933, Army Research Office grant W911NF2110095, and generous donations from corporate partners. Paul Zhang acknowledges the support of the Department of Energy Computer Science Graduate Fellowship and the Mathworks Fellowship. The authors thank Justin Solomon and David Bommes for many valuable discussions.

## References

- [1] Weingarten V.I. “The controversy over hex or tet meshing.” *Machine design*, vol. 66, no. 8, 74–76, 1994
- [2] Cifuentes A., Kalbag A. “A performance study of tetrahedral and hexahedral elements in 3-D finite element structural analysis.” *Finite Elements in Analysis and Design*, vol. 12, no. 3-4, 313–318, 1992
- [3] Benzley S., Perry E., Merkley K., Clark B., Sjaardema G. “A Comparison of All Hexagonal and All Tetrahedral Finite Element Meshes for Elastic and Elasto-Plastic Analysis.” *International Meshing Roundtable*, vol. 17, 01 1995
- [4] Ray N., Sokolov D., Lévy B. “Practical 3D frame field generation.” *ACM Transactions on Graphics*, vol. 35, no. 6, 1–9, Nov. 2016. URL <http://dl.acm.org/citation.cfm?doid=2980179.2982408>
- [5] Lyon M., Bommes D., Kobbelt L. “HexEx: Robust Hexahedral Mesh Extraction.” *ACM Trans. Graph.*, vol. 35, no. 4, 123:1–123:11, Jul. 2016
- [6] Nieser M., Reitebuch U., Polthier K. “CubeCover–Parameterization of 3D Volumes.” *Computer Graphics Forum*, vol. 30, no. 5, 1397–1406, Aug. 2011. URL <http://doi.wiley.com/10.1111/j.1467-8659.2011.02014.x>
- [7] Huang J., Tong Y., Wei H., Bao H. “Boundary aligned smooth 3D cross-frame field.”

- SIGGRAPH Asia*, p. 1. ACM Press, Hong Kong, China, 2011. URL <http://dl.acm.org/citation.cfm?doid=2024156.2024177>
- [8] Solomon J., Vaxman A., Bommes D. “Boundary Element Octahedral Fields in Volumes.” *ACM Transactions on Graphics*, vol. 36, no. 3, 1–16, May 2017. URL <http://dl.acm.org/citation.cfm?doid=3087678.3065254>
  - [9] Li Y., Liu Y., Xu W., Wang W., Guo B. “All-hex meshing using singularity-restricted field.” *ACM Trans. Graph.*, vol. 31, no. 6, 177, 2012
  - [10] Jiang T., Huang J., Wang Y., Tong Y., Bao H. “Frame Field Singularity Correction for Automatic Hexahedralization.” *IEEE Trans. on Visualization & Computer Graphics*, vol. 20, no. 8, 1189–1199, Aug. 2014
  - [11] Liu H., Zhang P., Chien E., Solomon J., Bommes D. “Singularity-constrained octahedral fields for hexahedral meshing.” *ACM Transactions on Graphics*, vol. 37, no. 4, 1–17, Jul. 2018. URL <http://dl.acm.org/citation.cfm?doid=3197517.3201344>
  - [12] Corman E., Crane K. “Symmetric Moving Frames.” *ACM Trans. Graph.*, vol. 38, no. 4, Jul. 2019. URL <https://doi.org/10.1145/3306346.3323029>
  - [13] Gao X., Panozzo D., Wang W., Deng Z., Chen G. “Robust structure simplification for hex re-meshing.” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, 1–13, 2017
  - [14] Gao X., Deng Z., Chen G. “Hexahedral mesh re-parameterization from aligned base-complex.” *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, 1–10, 2015
  - [15] Xu G., Ling R., Zhang Y.J., Xiao Z., Ji Z., Rabczuk T. “Singularity Structure Simplification of Hexahedral Meshes via Weighted Ranking.” *Computer-Aided Design*, vol. 130, 102946, 2021
  - [16] Campen M., Shen H., Zhou J., Zorin D. “Seamless parametrization with arbitrary cones for arbitrary genus.” *ACM Transactions on Graphics (TOG)*, vol. 39, no. 1, 1–19, 2019
  - [17] Peng C.H., Zhang E., Kobayashi Y., Wonka P. “Connectivity editing for quadrilateral meshes.” *Proceedings of the 2011 SIGGRAPH Asia conference*, pp. 1–12. 2011
  - [18] Ledoux F., Shepherd J. “Topological modifications of hexahedral meshes via sheet operations: a theoretical study.” *Engineering with Computers*, vol. 26, no. 4, 433–447, 2010
  - [19] Bracci M., Tarini M., Pietroni N., Livesu M., Cignoni P. “HexaLab.net: An online viewer for hexahedral meshes.” *Computer-Aided Design*, vol. 110, 24–36, 2019. URL <https://www.sciencedirect.com/science/article/pii/S0010448518304238>
  - [20] Quadros R. “The CUBIT Geometry and Meshing Toolkit.” <https://cubit.sandia.gov/> 2021
  - [21] Karčiauskas K., Peters J. “Minimal bi-6 G2 completion of bicubic spline surfaces.” *Computer Aided Geometric Design*, vol. 41, 10–22, 2016
  - [22] Karčiauskas K., Peters J. “Refinable smooth surfaces for locally quad-dominant meshes with T-gons.” *Computers & graphics*, vol. 82, 193–202, 2019

# INCREMENTAL DECOMPOSITION FOR HEX-MESHING IN CAD USING VIRTUAL TOPOLOGY

Benoit Lecallard<sup>1</sup>, Trevor T. Robinson<sup>1</sup>, Cecil G. Armstrong<sup>1</sup>, Declan C. Nolan<sup>1</sup>, Harsha Ramesh<sup>2</sup>

<sup>1</sup>*Queen's University Belfast, Belfast, United Kingdom. b.lecallard@qub.ac.uk*

<sup>2</sup>*Rolls-Royce plc, Derby, United Kingdom Harsha.Ramesh@rolls-royce.com*

## ABSTRACT

This paper presents methods for preparing a geometry model for finite element mesh generation in a Mechanical Computer-Aided Design (MCAD) environment. It works by creating a new representation of the model through the application of virtual topology operators. The resulting “analysis topology” description is used to abstract the analysis model, enabling automated tools and experts to apply an incremental strategy to decompose the model for meshing, without modifying the original CAD model. This work also demonstrated how virtual topology enables the integration of multiple model decomposition tools to expand the capabilities of the hosting CAD environment, providing support for more meshing strategies and more freedom in how they are applied, while bridging the gap between the CAD and analysis models. Herein, the virtual topology operators used to decompose the model are checked and propagated based on the required mesh constraints to ensure the resulting mesh is conformal at the interfaces. Finally, the methods required to decompose the original CAD model using the analysis topology description and “virtual geometry curves” are presented, enabling downstream automation of the mesh.

**Keywords:** mesh generation, analysis topology, virtual topology, CAD

## 1. INTRODUCTION AND RELATED WORK

Generating a good quality mesh is a major bottleneck in most finite element analysis workflows. The generation of high-quality hexahedral (Hex) element meshes remains a highly skilled and user intensive task, which often requires the use of dedicated CAE packages into which the original CAD geometry needs to be transferred from the CAD environment. Hex elements are preferred over alternatives (e.g. tetrahedral elements) when simulating highly non-linear events, using explicit analysis codes and for accurate contact capture between deformable bodies. A comprehensive survey by Sarrate et al. [1] highlights a wide range of approaches to hex meshing, as well as the benefits of using this element type. Decomposition-based approaches are widely used and involve partitioning the geometry of the model to be meshed into sub-regions with specific topological and shape characteristics which can be meshed using hex-meshing algorithms like mapping and sweeping.

With decomposition and meshing accounting for more than 50% of the time taken for the entire simulation task [2], automating aspects of the hex meshing task is a well-researched ambition. Whilst the push toward fully automated

hex-mesh generation for arbitrary domains has yet to yield a generic solution, it has resulted in many automated tools that are applicable to specific classes of geometry. These tools use either divide and conquer paradigms to recursively extract simple regions [3]–[6], or use intermediate constructs to capture the flow of elements and identify partitions [7]–[10]. An extension of this is to apply the same divide and conquer paradigms in an integrated incremental decomposition workflow, where simpler tools alleviate the task of the more complex and computationally expensive ones. While analysts would greatly benefit from combining existing tools, their integration is challenging as standards for geometry exchange are not tailored for analysis models. Dedicated meshing packages such as CUBIT[11] already integrate various automated methods, but still resort to the judgment of the user to select the best partitioning strategy. These packages are also limited by the need to transfer the geometry from a CAD environment, and the difficulty to add additional decomposition methods. The shortcomings of fully automatic tools are also recognized in [12], where the benefits of semi-automated decomposition workflows are demonstrated using a manual sketch-based decomposition method enhanced by geometric reasoning [13].

When a model is decomposed into sub-regions, a conformal mesh is required at the interfaces to successfully connect their respective meshes. The constraints of conformal hex meshing and structured mesh implications are reviewed by Blacker [14]. Previous work on generating conformal meshes using sweeping is described in [15] and [16]. Even though both are mesh-based methods tailored for one type of decomposition, they highlight the importance of interface management for conformal meshing.

The benefit of using virtual topology for pre-processing a model for meshing has been presented by Sheffer et al. [17]. The concept involves creating virtual topology entities by applying virtual topology operators to the entities in the original CAD model, which are therefore based upon but do not alter the underlying CAD definition. To date it has mostly been used for correcting minor “defects” (e.g. to merge a sliver face with a larger adjacent face), with implementations focused on the final steps of the analysis model preparation process. Extending the use of virtual topology to the entire pre-processing stages would facilitate the integration of different automated tools, as the need to exchange geometry (e.g. decomposed CAD) and/or pre-processing operations (e.g. split operation) is replaced by the need to exchange virtual topology operations. White [4] used virtual decomposition to automate hex mesh generation, where surface nodes of an initial mesh are reassigned to a virtual sub-region.

More recently, Tierney et al. used virtual topology operators to generate an “analysis topology” based on the outputs of a decomposition algorithm [18]. The concept of analysis topology enables to streamline pre-processing tasks, by adding flexibility to the decomposition while exposing all the necessary information to manage interfaces and automate decomposition and meshing. However, the implementation in that work was limited by the need to edit automated tools to work using virtual topology, and the a-posteriori identification of meshing strategies preventing further decomposition in the absence of a mechanism to maintain a conformal mesh at interfaces. Finally, generating a mesh from a virtually decomposed model requires either new meshing tools or robust geometrical decomposition capabilities for compatibility with existing meshing tools.

This work builds on the analysis topology concept to enable incremental decomposition of CAD models for automatic hex meshing. The main contributions include introducing a method to integrate virtual topology with both existing tools and manual operations and a method to manage and exploit meshing strategies to propagate splits automatically. Finally, a method to ensure that the virtual topology decomposition can be applied geometrically for compatibility with downstream meshing is presented.

## 2. INCREMENTAL DECOMPOSITION FOR MESHING

Incrementally decomposing a model for meshing involves identifying and extracting individual regions of the geometry to which known meshing algorithms can be applied. Once a meshing strategy has been identified for each sub-region of the domain, each can be meshed in a piecewise manner.

### 2.1 Structured meshing requirements

The quality of a hex mesh is directly related to the geometry of its elements and their connectivity. In a regular mesh each interior node should connect exactly 4 quad elements or 8 hex elements. To accommodate complex shapes while retaining the quality of individual elements, nodes must sometimes connect an irregular number of elements, which introduces “singularities” into the structure of the mesh.

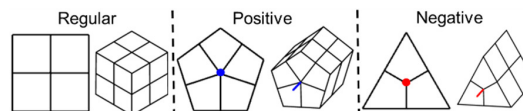


Figure 1. Mesh singularities.

**Definition:** A **mesh singularity** is a collection of one or more irregular nodes. It can be either positive (more than the regular number of connected elements), or negative (less than the regular number of connected elements), as shown in Figure 1.

Quad (2D) and hex (3D) meshing algorithms impose strict requirements on the presence of singularities, which in turn impose constraints on the shapes that can be processed, as the number of singularities is directly linked to the shape. These requirements are as follows:

- **Mapping** (quad): The mesh is generated by mapping the template of a unit square onto a local surface parametrization [19]. As such, no singularities will occur and the face must be 4-sided. It also implies that opposite pairs of edges need to have the same number of divisions. A sub-mapping variant is also possible for non-rectangular faces where all the edges can be grouped into two sets of opposite groups.
- **Paving** (quad): The mesh is generated by inserting rows of quad elements from the boundaries towards the interior [20]. There are no specific requirements on the structure of the mesh and singularities can be present. This means there is no constraint on the shape of the face. However, the algorithm may introduce pairs of singularities that cancel each other. It also requires that the sum of division numbers on each loop of edges must be even.
- **Mapping** (Hex): The mesh is generated by mapping the template of a unit cube on the local i-j-k parametrization. This requires the shape to have a cube-like topology, with 6 logical faces and 12 logical edges, and no singularities can be present. It implies that all bounding faces are mapped meshed, with the associated constraints on singularities and edges divisions.
- **Sweeping** (hex): Hex elements are generated by sweeping quad elements on a **source face** to a **target face**. This means all lateral faces (so-called **wall faces**) connecting the source and the target have mapped mesh structures, and hence no singularities can exist on wall faces. Also, corresponding edges at opposite ends of the sweep must have the same number of divisions. There is no mesh structure requirement on the source face, which can be either

mapped or paved, and therefore singularities can be channeled from the source to target face.

This work focuses on two types of shapes suitable for hex meshing with these algorithms:

- **Block** shapes, with a cube-like topology that can be map (Hex) meshed with 6 faces mappable (quad).
- **Sweepable** shapes, with a loop of mappable (quad) wall faces in the sweep direction.

Directly identifying a block decomposition for an arbitrary geometry is difficult, as there should be no singularities in the blocks. This means all singularities need to be located at the edges bounding the interfaces between blocks. Sweepable regions are less constraining as they can accommodate singularities along the sweep direction and are therefore easier to identify. There is a strong correlation between the two types, as a block can be swept meshed in any of three directions, and sweepable regions can have a mappable source face and therefore satisfy block constraints. It is therefore easier to identify first a semi-structured mesh by identifying sweepable regions, and then decomposing their source faces to constrain the singularities and achieve a more structured block decomposition.

## 2.2 Reasoners

Manually identifying and extracting block and sweepable regions can be a very tedious task for geometries which include many details. Various automated tools or reasoners have been developed to facilitate this task by extracting regions based on specific geometric and topological characteristics. These characteristics define in turn a meshing strategy which specifies how the regions should be meshed. This information is required as the type of the shape (block or sweepable) does not contain sizing information and can change.

**Definition:** A **decomposition reasoner** refers to an algorithm that queries the model to identify regions that can be assigned a specific hex-meshing strategy and provides the topological and geometrical information to create the partitioning entities necessary to extract such regions.

**Definition:** A **meshing strategy** describes the type of element (e.g. Hex or Mixed-Tet) along with sizing information, symmetries and anisotropic element shape metric properties of the region.

The simplest reasoners are tools that identify regions that are already blocks or are sweepable, by checking that the topology and geometry match the requirements of that region type (described previously). Other reasoners use shape properties such as concavities and symmetries to help breaking down a model into simpler regions. For example, aero-engine models are mostly axisymmetric with cyclic patterns that repeat around the circumference. Using a dedicated reasoner based on [21], axisymmetric regions and regions that can be meshed using cyclic symmetries can be identified. The associated meshing strategy stores any repetition pattern, to ensure a compatible mesh between each occurrence. Other reasoners exploit local anisotropy of the shapes to identify sweepable regions. For example, thin-walled regions with two large dimensions compared to the

third can be meshed by applying a mesh to a larger face and sweeping through the small thickness. A thin-sheet reasoner based on Sun's implementation [22] identifies and extracts thin regions by manipulating pairs of opposing faces from the CAD geometry. The associated meshing strategy stores the aspect ratio of the shape and the thickness, which can then be used to infer a target element size as described in [23]. Similarly, truss-like structures, or models which have had their thin-sheet regions removed, can have many long regions with a nearly constant cross-section topology, that are also appropriate for hex-meshing by sweeping. These can be identified by a long-slender reasoner that processes loops of nearly parallel long edges, as described by Sun [5]. These reasoners can greatly reduce the number of DOFs of the mesh, as the anisotropy of the region can be used to stretch the hex elements and reduce their number. More complex decomposition reasoners can also make use of other types of information, such as temporary constructs (frame-fields, medial-object), functional and adjacency information if available, or AI methods.

Each reasoner has its strengths and weaknesses in terms of speed, accuracy and class of shapes supported. More than one may be required to achieve a full hex mesh for a complex shape. Therefore, an efficient incremental decomposition workflow requires the integration of a diverse range of decomposition reasoners. To be of maximum benefit these need to work in any order, without any dependencies on the preceding reasoners or the package where the CAD model is hosted. Preparing a CAD model for meshing can also include de-featuring and dimensional reduction operations, which can be identified and applied using dedicated automated reasoners which are not covered in this paper.

## 2.3 Challenges

Since many meshing workflows start from a geometry that has been created in a feature-based CAD environment, and to maintain the associativity with the design history in the model, the ability to decompose the model for meshing within the CAD system is an attractive solution. However, there are several challenges to doing so, primarily because CAD packages have not been developed for the purposes of decomposing a model for meshing.

First, creating a split operation in CAD may create unexpected geometrical defects such as sliver faces and result in non-watertight models due to trimming errors [24]. Secondly, automating the decomposition and downstream meshing requires a robust tracking of B-Rep entities, which is challenging due to persistent naming issues inherent to CAD packages [25]. Then, incrementally decomposing the CAD model will append a sequence of split operations to the feature tree of the model, and any edit further up in the tree may produce unexpected results further down, including the splits. Finally, most commercial CAD environments rely on a manifold boundary representation scheme, meaning that two bodies cannot share a same face, edge or vertex. Hence, two identical faces are created within the CAD system at the interface between two bodies after a split operation.

Even when a CAD system is used to help prepare a geometry model for meshing, a transfer to a dedicated CAE package is usually still required for meshing. After doing so the



decomposition will be converted to a non-manifold representation which ensures the resulting mesh is conformal at interfaces between regions. It is therefore important to ensure that incremental decomposition will produce a usable collection of bodies that can be re-assembled in a CAE package for meshing.

Another challenge comes from the incremental decomposition principle itself. Identifying simple regions first means all of the complexity of the meshing task will be pushed to the last regions of the geometry to be processed. This can become problematic as these regions may harbor complex arrangements of singularities. Where these exit through an interface, they make any hex meshing strategy in connected regions invalid. Therefore, special care must be taken when chaining reasoners, as structure modification can propagate throughout the decomposition.

## 2.4 Proposed workflow

Since most of the challenges of incrementally decomposing a model in CAD come from the application of the successive split operations, the idea in this work is to identify regions to which a known meshing strategy can be applied, store the required partitioning strategy, and then query the partitioning strategy to identify the next regions to process. This is enabled by virtual topology split operators that will topologically partition the model without altering the CAD representation, as described in the next section. Each region in the model for which a meshing strategy has not yet been identified is classed a “residual region”. Eventually, once all the reasoning is done and a suitable virtual topology decomposition is available, the model can be decomposed within the CAD system to be used for meshing. Should any residual regions remain at the end of the process a tet-mesh can be applied to them, with a layer of pyramid elements at interfaces with hex-meshed regions, to produce a mixed mesh. To be successful, this workflow requires a simple way of integrating existing reasoners with virtual topology, so they can identify suitable regions in presence of virtual topology and define virtual topology splits. The meshing strategies identified by the reasoners need to be robustly managed to remain valid after further decomposition of neighbor regions. Finally, the ability to robustly convert a virtual decomposition into a CAD decomposition is required to ensure the virtual decomposition is usable.

## 3. ANALYSIS TOPOLOGY

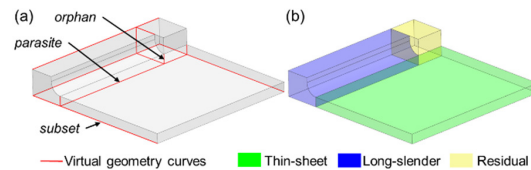
### 3.1 Virtual topology

Virtual topology uncouples the topological representation of a model from its geometrical representation in the B-Rep scheme [17], allowing manipulation of the topology without having to alter the underlying geometry of the model. It defines a set of entities and operators to carry out the operations associated with model pre-processing for meshing, and to formalize the relationships with the original host model.

Virtual topology entities do not require an explicit geometric definition and instead use a geometric definition inferred from their host entities, or which can be related to simple

geometrical constructs (e.g. line between two points, least-square fitted surface, etc.). These are illustrated in Figure 2 (a), and include:

- **Parasite entities:** entities that do not exist in the topology of the original CAD model, but lie on an entity from the original CAD model of higher dimension (e.g., an edge lying on the face it splits).
- **Subset entities:** subsets of host entities that are split by a topological entity of lower dimension (e.g., faces obtained by partitioning a host face with a parasite edge).
- **Superset entities:** a superset of host entities that are merged together by ignoring their common boundary entities.
- **Orphan entities:** an entity without a host one dimension higher, and from which no geometry description can be inherited (i.e., an edge through volume).



**Figure 2. (a) virtual topology entities created after virtual decomposition and (b) equivalent geometric decomposition and meshing strategies.**

Virtual topology operators relevant for an incremental decomposition workflow are the virtual topology split, where a host entity is split into several subsets by parasite entities, and the virtual topology merge where several entities of the same dimension are merged into a single superset by ignoring their common boundary entities.

### 3.2 Abstracting the analysis model

**Definition:** The **analysis model** is a transformed version of the design model that exists within a CAE environment, to which mesh, boundary conditions and loads are applied.

Implementing the decomposition in the CAD system using virtual topology operators means that only a topological description of the analysis model is created, known as the analysis topology.

**Definition:** The **analysis topology** is a representation of the boundary topology of the analysis model.

In this work, the analysis topology is a non-manifold cellular model, which means that all interfaces between cells are known and are considered cells in their own right. Meshing strategies can be attached to cells. The analysis topology is initialized by extracting the topological representation of the B-Rep from the original design model. It is external to any CAD package and can represent topological relationships not supported in many CAD environments, but which are required for conformal meshing. It is therefore capable of acting as the interface between different CAD and CAE



packages. However, while the analysis topology can be used to represent the topology of the model to be meshed, it does not contain sufficient information to be used for reasoning. To address this issue, “virtual geometry” is introduced.

**Definition:** **virtual geometry** entities are geometric representations of virtual entities that co-exist in the modelling space of the design model, but are not associated with its B-Rep.

Virtual geometry entities are used to perform geometric tests on the analysis topology and to visualize the virtual volume cells. Virtual geometry curves (in red in Figure 2 (a)) are combined with the existing edges of the CAD model that have not been virtually edited to define a wireframe representation of the volume cells. These curves help store the partitioning intent of decomposition reasoners and avoid deleting and re-creating curves. Whenever the actual CAD decomposition is required, virtual geometry curves are used to define virtual geometry surfaces that can partition the CAD model to generate the equivalent analysis model, Figure 2 (b).

### 3.3 Reasoning on the analysis topology

The use of an analysis topology implies that the current decomposition state of a model is not explicitly available and cannot be directly queried or decomposed. Additional steps are required to adapt the decomposition reasoners, which depend on the ability to integrate a reasoner. Figure 3 shows the integration of 6 different types of reasoners to interact with the analysis topology

#### 3.3.1 Queries

Reasoners that are fully integrated with virtual topology can directly query the analysis topology. Geometrical queries are achieved by inheriting the geometric definition of host entities or by querying virtual geometry curves if no geometry is linked. Reasoners for extracting thin-sheets, long-slender and axisymmetric regions have been fully integrated with virtual topology, as described in [26]. Other reasoners that are not integrated with virtual topology require an explicit geometry description to work with, as modifying their implementation to work with virtual

topology might be tedious, or not even possible. In that case, there is no need to commit the entire decomposition, only the subset regions of interest can be temporarily extracted from the CAD model, as detailed in section 5.1. The temporary region can then be processed in either another CAD session of the native CAD environment, or a different CAD environment after STEP export.

In the situation where a user wants to manually insert partitions by applying CAD split operations, an explicit geometry is also extracted. It is then enriched with interface and mesh singularity information from neighbors, to help the user understand the flow of elements and constraints stemming from the meshing strategies of neighbor regions. (see Figure 13 (c)).

#### 3.3.2 Parasite wireframe

In the absence of any standard for exchanging virtual topology partitions (though one could easily be defined), the concept of a parasite wireframe is introduced to integrate the output of different reasoners, or manual intervention, around a common format. The purpose is to collect the minimal information required for applying virtual topology split operators that cannot be recovered by reasoning, to accompany the transfer of the geometry as a STEP file.

**Definition:** A **parasite wireframe** is a collection of vertices, curves and loops of curves that represent virtual topology parasite vertices, parasite edges and parasite faces respectively.

Additional information can also be included to reduce processing time, such as host entity information for each vertex and curve to establish the link with the model to decompose, the bounded/bounding relationship between vertices and curves and which operation can be used to recreate a face from the loop of curves (e.g., swept surface, fill surface). Since the objective is to apply a virtual topology split, and the final position of the nodes on these faces may eventually depend on a mesh smoothing algorithm, the exact geometry of the partition is not required. Hence, transferring the CAD curves only is sufficient and it is more flexible to transfer the scaffold required to define the cut faces than the cut faces themselves.

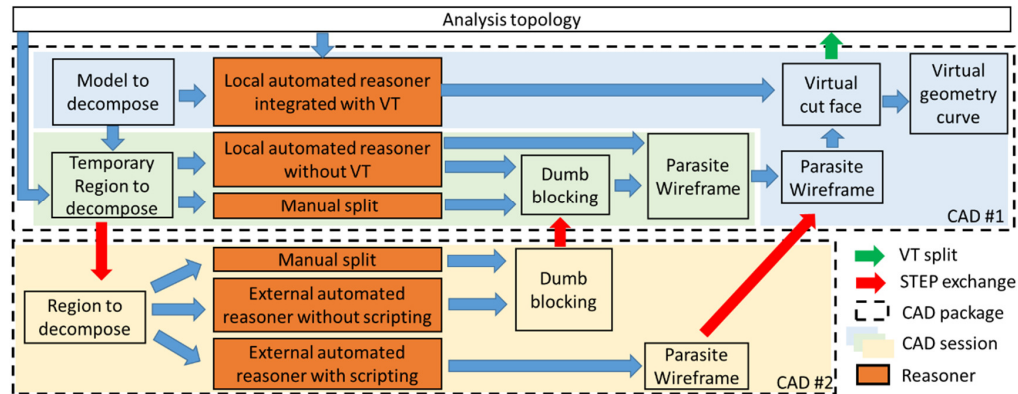


Figure 3. Integration of different reasoners with virtual topology.

### 3.3.3 Output processing

Reasoners that are already fully integrated with virtual topology directly define virtual topology splits and produce the necessary virtual geometry curves. For other reasoners, with some level of scripting available, defining a simple parasite wireframe is straightforward. It can then be transferred back to the CAD session of the original CAD model and processed to define virtual topology splits. The curves contained in the parasite wireframe can be used directly as virtual geometry curves or can be reconstructed to get a better fitting with the CAD model. A user can also directly specify a parasite wireframe, although it can be tedious as the curves forming a loop of a face need to be grouped manually.

For reasoners that only output a CAD decomposition, or after the user is done splitting the region of interest, the dumb blocking that results is converted into a parasite wireframe using an automated routine. It first queries all the edges and faces of each block, to identify and match coincident entities stemming from the manifold nature of the splits. Then entities are classified as existing, subset or parasite entities by comparing them with the entities of the region of interest before splitting, that are matching the analysis topology. Only parasite entities are kept to define the parasite wireframe and their host entity is also recorded. This parasite wireframe is then transferred to the original CAD environment to define the virtual topology splits.

## 4. SPLIT PROPAGATION

In the analysis topology, each face of each body has its own meshing strategy assigned, which is inferred from the meshing strategy of the parent body or bodies in the case of an interface. Whenever the topology of a face is modified to accommodate imprints, either to decompose the face or because of further decomposing neighbor regions sharing the interface, the flow of elements or the net number of singularities on the face may change. This implies that decomposing a body to extract hex-meshable regions can invalidate the meshing strategies previously identified on adjacent regions. As a result, special care must be taken to maintain meshing strategies as the model is incrementally decomposed.

### 4.1 Imprints and interfaces

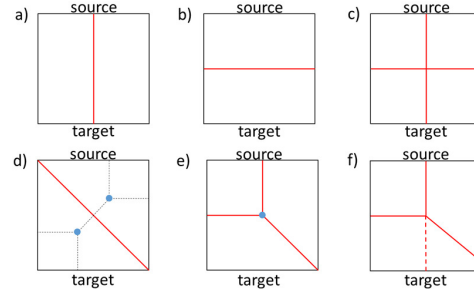
Each face of the analysis topology is assigned one of the following meshing strategies:

- unstructured triangular mesh: this only exists on or between residual regions.
- unstructured quad mesh (e.g., paved): on source and target faces of sweepable regions.
- structured quad mesh (e.g., mapped): faces of block regions, walls of sweepable regions.

Unless it is an interface with a hex meshed region, there is no limitation on partitioning the faces of residual regions. In the case of source and target faces, singularities can be channeled, therefore there is no limitation on partitioning these faces. However, doing so may transform a simple one-to-one sweep into a many-to-many sweep that is not supported by many

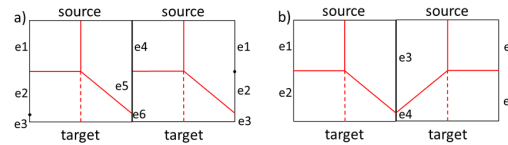
meshing tools. The condition on mapped interfaces is the most stringent, as it implies that the result of a split/imprint on the face must be a collection of faces with the same mappable properties. Otherwise, the sweepable/block strategy of the bodies will become invalid and reprocessing will be needed.

The validity of an imprint on mapped interfaces is assessed by checking how it modifies the flow of elements associated with the interfaces. The direction of the flow of elements is only modified by the introduction of negative or positive singularities on the face, which either stem from a subset with a non-null net singularity number, or from the topology of the imprint itself. Figure 4 shows various imprints on a wall face of a swept region (which must be 4 sided). The imprints in Figure 4 (a), (b) and (c) do not perturb the flow of elements from top to bottom and left to right, so they are valid, and the body bounded by the face is still sweepable. The imprints in Figure 4 (d) introduce two triangular faces that would require negative singularities (in blue). In Figure 4 (e), while all the subset faces are 4 sided, the connectivity of the imprints introduce a negative singularity that redirects part of the top-down flow of elements to the left. Figure 4 (f) is inconclusive when considering the bottom subset as a logical rectangle, as all the subsets have 4 corners and are mappable.



**Figure 4. Valid imprints (a-c) do not modify the mesh flow, (d-e) introduce singularities making the sweep invalid, and (f) is inconclusive.**

Even if all the imprints on all individual wall faces are valid and only result in mappable faces, sweepable regions require that the wall faces form a loop of mappable faces. This introduces an additional constraint on the flow of elements, which is assessed by solving the mapping constraint on the number of elements. In Figure 5, two mappable faces forming a loop receiving valid imprints are laid flat. In Figure 5 (a), solving the equality constraint on opposite edges yields  $Ne_2 = Ne_5 = 0$  (where  $Ne\#$  is the number of element edges on edge  $e\#$ ) which implies that the loop cannot be meshed unless the imprints are moved. On the other hand, the configuration in Figure 5 (b) is valid, but will result in elements being stretched on  $e_3$  and compressed on  $e_4$ .



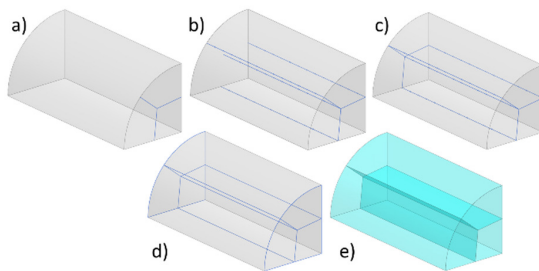
**Figure 5. Loop of mappable faces with (a) invalid and (b) valid edge division balancing.**

Block topology is a special case of sweepable regions with 3 pairs of opposite faces resulting in 3 possible sweeping axes, and all faces mappable. Therefore, the same approach for checking invalid imprints can be used. The only difference is that some invalid imprints can be handled by reclassifying the shape type from block to sweepable, provided there is still a loop of valid mappable faces.

If the imprints are valid, the decomposition of the face or volume can go ahead. If the face is an interface the question of the propagation of the split arises. For single imprints on wall faces aligned with the sweep as shown in Figure 4 (a), there is no need to propagate the imprint as all the wall faces of the sweep remain 4-sided.

#### 4.2 Aligned split

The process of splitting a sweepable region by propagating imprints along the sweep direction is illustrated in Figure 6 (a), where a sweepable body has had its source face imprinted to match quad meshing requirements (in this case, imprints have been created by mid-point decomposition reasoner applied to the face). A new parasite wireframe is created to store the split information. The curves of the imprint are added along with their host face, and vertices are processed to identify host curves and merge coinciding ones. Wall edges are discretized and are used to trace discretized curves aligned with the sweep on wall faces and inside the volume, as shown in Figure 6 (b). Curves that are lying on a wall face are re-projected if an explicit surface is available, and all the curves are added to the parasite wireframe. Finally, the curves matching the imprint curves on the opposite target face are created by joining the last points of the newly created curves to match the topology of the imprint. This completes the parasite wireframe with one loop of curves identified for each imprint, producing 3 parasite faces, as shown in Figure 6 (c). The resulting analysis topology after virtual topology split and the equivalent geometric decomposition are shown in Figure 6 (d) and (e) respectively, with three simple sweepable regions without imprint generated.

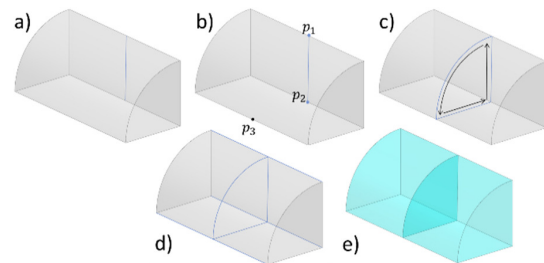


**Figure 6. Imprints on the source face are propagated along the sweep direction to create virtual parasite faces splitting the sweepable region into 3 parallel sweepable regions.**

#### 4.3 Perpendicular split

Since sweepable regions are defined by a loop of mappable wall faces around the sweep direction, the propagation of imprints that are perpendicular to the sweep direction is achieved by exploiting mapping constraints to trace loops of

curves. The resulting curves partition the loop of wall faces into two or more loops of mappable faces, effectively splitting the original sweep region into a chain of sweepable regions, as described in Figure 7. As for the propagation of aligned splits, a new parasite wireframe is first created and the imprint curves on wall faces (Figure 7 (a)) are added. Then, all coincident vertices are merged, and the parameter of each vertex lying on a wall edge is extracted. These parametric values are clustered within a tolerance range and new vertices are created for each cluster on wall edges without a vertex using the mean value of the cluster. In Figure 7 (b), vertices with parameters  $p_1$  and  $p_2$  are clustered, and a new vertex with parameter  $p_3$  is created. Once all vertices are created, the loop of wall edges is traversed for each cluster, and vertices without existing parasite curves are joined by tracing a new curve on the wall face. The resulting loop of curves are added to the parasite wireframe and used to define a parasite face, as shown in Figure 7 (c). The resulting analysis topology after virtual topology split and the equivalent geometric decomposition are shown in Figure 7 (d) and (e) respectively, with a chain of two simple sweepable regions without imprint generated. This algorithm enables processing of multiple imprints on multiple wall faces and to propagate cuts on wall edges only.



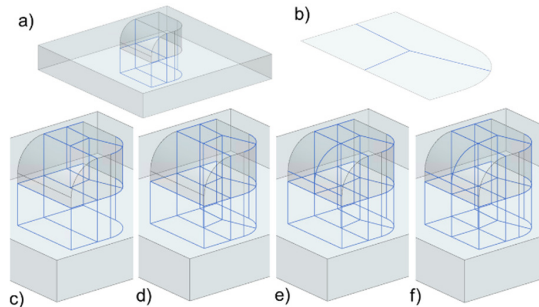
**Figure 7. Perpendicular imprint on a wall face is traced around the loop of mappable wall faces to create a virtual parasite face splitting the sweepable region into 2 stacked sweepable regions.**

#### 4.4 Identifying propagation order

As the model is incrementally decomposed, the number of hex meshable regions increases throughout the process and their interaction becomes more complex. Since propagating imprints to partition sweepable bodies also produces new imprints on adjacent bodies, special care must be taken when propagating splits. If the meshing strategies assigned result in a valid mesh, propagating the imprints following the meshing constraints will also produce a valid mesh. As such, the order in which imprints are propagated in the sweep direction and perpendicular to it does not matter. However, since imprints on source faces can modify the number or position of singularity lines, it is better to propagate aligned splits first, to ensure proper channeling of the singularities.

In Figure 8 (a), the model is decomposed into one thin region and 4 sweeps. The source face of one sweepable region is decomposed resulting in the imprints in Figure 8 (b), which are first propagated to split the region (Figure 8 (c)) introducing both perpendicular and aligned imprints on neighbor sweeps. The imprints on the source faces are processed first (Figure 8 (d)), followed by the lateral propagation (Figure 8 (e)).

Eventually, the last sweep has compatible imprints on both its wall face and source face, which are propagated in the sweep direction, Figure 8 (f)).



**Figure 8. Imprints on the source face (b) are first propagated to the connected sweepable body (c) resulting in new imprints on neighbor regions that are recursively propagated (d-f) until no more splits can be found on sweepable bodies.**

In some cases, additional meshing constraints stemming from symmetry properties and patterning can arise, where not only the topology but also that actual geometry must be matching between faces to reconnect everything. This is handled by applying the symmetry/patterning transform to the imprint curves before propagating them, to ensure they are correctly located.

## 5. DECOMPOSITION IN CAD

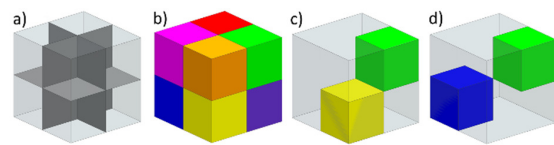
Once the incremental decomposition is complete with all the splits correctly propagated, and when no more hex meshable regions can be identified, the virtual volume cells can be extracted to generate a meshable analysis model. Rather than trying to apply a sequence of split operations matching the virtual topology operators applied, the model is decomposed by querying and using all the interfaces between bodies as cutting faces. This provides a more flexible way of partitioning the model that does not rely on the history of the decomposition process, while allowing a single region to be extracted in the model without having to perform the entire decomposition.

The final analysis model must be contained within a non-manifold CAE environment to ensure a conformal mesh is created at interfaces. The partitioning of the geometry can either be applied in a CAD environment or a CAE environment. In the first case, the virtual geometry curves are used to create the cutting surfaces, and the final blocking is exported to the destination meshing environment. In the second, virtual geometry entities are exported, and the model is decomposed by applying split operations through an API.

If the geometry decomposition is performed in a non-manifold environment the process is straightforward, and the topology of the resulting analysis model will exactly match the analysis topology. If the decomposition is carried out in a manifold CAD environment, the limitations from the manifold representation and the split capabilities of the CAD engine must be taken into consideration.

### 5.1 Split ordering

Extracting all the subset regions identified in a single split operation has a high chance of failing in current tools, even for reasonably simple splits such as decomposing a cube into 8 octants (Figure 9 (a) and (b)). For this reason, an incremental decomposition approach is preferred, extracting regions of interest one after the other. This however produces intermediate bodies that can exhibit invalid non-manifold touch configurations even though all the final extracted bodies would be valid manifolds. In Figure 9 (c), if the green octant is removed first, extracting the yellow octant would create a non-manifold edge on the intermediate body (in translucent grey), hence the extraction would fail. Similarly, in Figure 9 (d), extracting the green octant first followed by the blue would create a non-manifold vertex on the intermediate volume.



**Figure 9. Invalid manifold condition on the intermediate body for different extraction order.**

This issue is eliminated by prescribing a decomposition order that avoids invalid intermediate volumes and maintains the manifold condition at all times. The process starts by querying all the internal vertex and concave edge neighborhoods to initialize the list of connected volumes. If the neighborhood is complete, e.g., a vertex is fully surrounded by geometry, any touching body can be removed. If the neighborhood is incomplete, e.g., a concave edge, the touching faces that are not interfaces define a front, and only bodies bounded by faces on that front are valid candidates for extraction. For each volume to be removed, the relevant neighborhoods are checked to ensure no touching condition will be created. If the extraction is valid the body is added to the decomposition sequence and the neighborhoods are updated. Else the candidate bodies are re-ordered before the current bodies and assessed in turn.

While this process results in a propagation of the partitioning front from the boundary, it also enables the extraction of a single region, by identifying the minimal number of regions that must be extracted first where the extraction would create an invalid intermediate volume. It also reduces the number of intermediate bodies, as these are difficult to manipulate since they do not match any volume cells in the analysis topology.

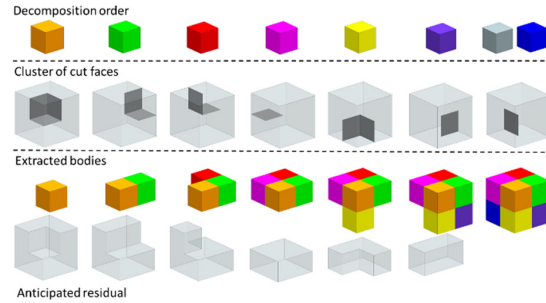
### 5.2 Cut definition

Once the order in which the regions need to be extracted is known, the sequence of split operations and cutting geometry required to perform the decomposition need to be generated. The cutting geometry is inferred from the interfaces between bodies recorded in the analysis topology. Virtual geometry curves are combined with the existing edges bounding each interface to generate a face by fitting a surface through the curves (in effect a fill surface operation). The resulting cutting faces are then clustered to match each successive split operation. This is achieved by querying all the interfaces of the



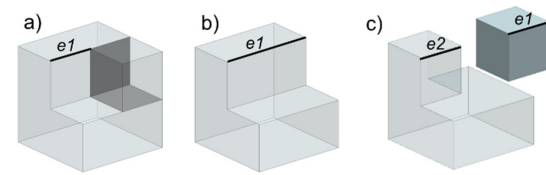
body to extract and removing the ones that have already been used. Adjacent faces with coincident edges are sewn together within each cluster.

shows the decomposition process for the model in Figure 9 (b). The first row shows the decomposition order identified, while the second row shows the different clusters of cutting faces generated for each split operation associated with this order. The third row shows the anticipated results from the incremental splitting, with all intermediate bodies being valid manifold representation in CAD.



**Figure 10. Cluster of faces identified for the extraction sequence. The intermediate body at each step is a valid manifold model.**

### 5.3 Subset mapping



**Figure 11. Persistent naming issue on edges.**

When automatically applying the sequence of split operations in a CAD package, special care must be taken at each step to identify which bodies need to be split and to remap entities on the subset corresponding to the region to extract. The remapping consists in matching the B-Rep entities that have been generated by the split operation with their topological analogue that already exists in the analysis topology. This is critical to ensure downstream automation of meshing but is made difficult by the way many CAD modelers implement split operations and how they suffer from the persistent naming problem. In Figure 11 (b), a common CAD practice is to merge faces that have the same underlying surface. As a result, the bold edge *e1* is extended to bound the merged face. When the merged faces are split to recover imprints or to extract the next region, one subset inherit the attributes of the parent, which may not match the original entity. In Figure 11 (c), the edge *e1* as moved to the right following the split.

When it comes to linking the representation in the CAD system with the analysis topology description, since the topology of the region being extracted matches the analysis topology it can be identified by looking first for the CAD bodies that have the same topology. If several CAD bodies are identified, the coordinates of the mid-point of the edges can be used to match

the correct subset. The re-mapping of the new CAD edges and faces is also recovered by matching the mid-point of edges.

When several intermediate bodies are created after a split, the host entity information is used to identify which one needs to be partitioned to extract the next region. All the faces and edges of the region to extract, that are subsets, are queried to get the list of host CAD entities. The intermediate CAD body that has the most matching CAD entities is then identified as the target for the splitting operation. If this test is not sufficient, point in volume methods are used to differentiate the bodies

Once all the regions have been extracted and remapped, a manifold collection of bodies will exist in the CAD environment, with all the coincident entities (e.g., bodies sharing a non-manifold interface in the analysis topology now have coincident faces in CAD) identified and labelled to automate the conversion to a non-manifold representation once transferred to a CAE package.

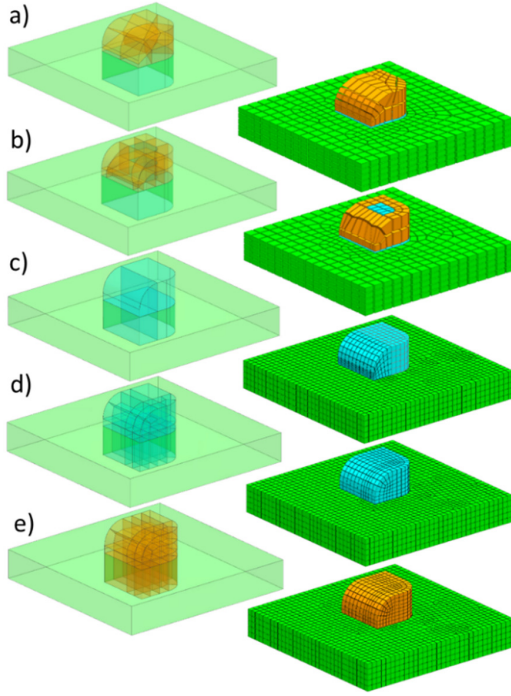
## 6. RESULTS

The incremental decomposition workflow is demonstrated within a virtual topology framework built around a relational database used to store the analysis topology, and the Siemens NX [27] CAD package, as described in [26]. In addition to various decomposition reasoners, the framework includes a meshing strategy reasoner to identify a meshing recipe from the meshing strategies. It uses integer programming to resolve mapping constraints and identify edge division numbers directly on the analysis topology. After the geometric decomposition is applied, another meshing reasoner is used to transfer the model to the NX CAE environment, recover the associativity with the analysis topology by merging coincident faces, and transfer the meshing recipe to automatically generate the mesh.

Within the current framework, fully automated workflows from the CAD model of the design to the mesh are only limited by the decomposition reasoners available and in identifying in which order they must be applied. In this work, this decision is left to the user, who applies the automated decomposition reasoners one after the other, and can also manually decompose the regions left by automated reasoners. Once satisfied with the analysis topology obtained, the user can adjust the meshing sizing parameters before the model is automatically decomposed geometrically and meshed. Further details on the virtual topology framework and automatic meshing are available in [26], and will be presented in a future paper.

### 6.1 Boss plate

Figure 12 presents different decompositions for a simple model of a plate with a boss that has fillets that introduce mesh singularities. All models are first processed using the thin-sheet decomposition reasoner, followed by a reasoner that identifies sweepable regions that are embedded in thin-sheets. In Figure 12 (a), a mid-point subdivision [28] reasoner is applied, resulting in a block decomposition but with all singularities meeting at the body mid-point. In Figure 12 (b), the residual is exported to CADFix [29] to use a reasoner based on the medial object.



**Figure 12. Different decompositions and meshes obtained for various combinations of decomposition reasoners and manual intervention.**

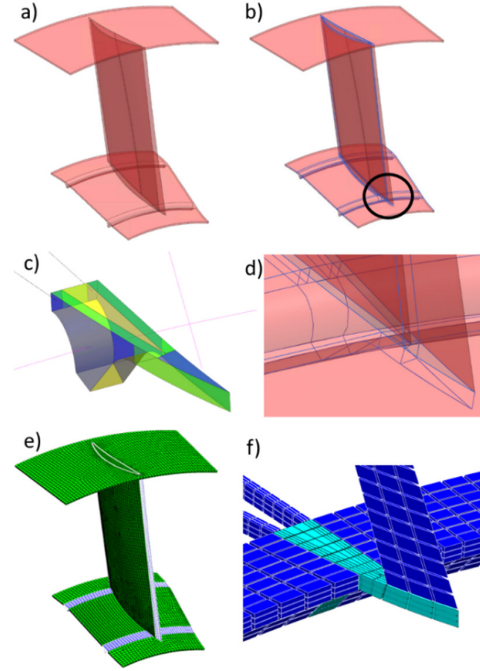
In Figure 12 (c), the user has specified a cutting plane to create two sweepable regions that channel the two singularity lines. In Figure 12 (d), 4 cutting planes are manually specified to extract sweepable regions, followed by automatic mid-point subdivision of the source faces to constrain the location of the singularity lines. In Figure 12 (e), the same manual decomposition is used but cube-shaped sweeps are reclassified as blocks, resulting in a full blocking of the residual.

## 6.2 Crescendo vane

Figure 13 shows the manual processing of a vane geometry to achieve a full hex mesh. After extracting symmetries and applying thin-sheet and long-slender tools (Figure 13 (b)), a complex residual region is left at the root of the leading edge. This is extracted (Figure 13 (c)) and manually partitioned into three sweepable regions, one to channel the singularity lines from the left and right long-slender regions, one to channel the singularity coming from the sharp leading edge, and one in between to channel the singularity coming from the yellow triangular face through the thickness. The operation is then converted into virtual splits resulting in Figure 13 (d), and the model can be automatically decomposed and meshed as seen in Figure 13 (e) and (f).

## 7. DISCUSSION

The robustness of the incremental decomposition based on virtual topology depends on several aspects. The robustness of the decomposition reasoners is not critical, as checks are



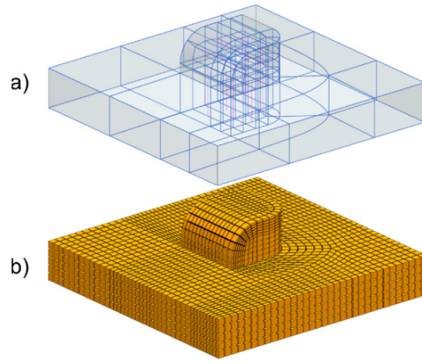
**Figure 13. (a) Crescendo vane model, (b) result of automated decomposition, (c) residual for manual processing, (d) manual split converted to virtual split and (e-f) resulting automated mesh.**

carried out after the regions have been extracted to ensure they are suitable for hex meshing. Failed reasoners can either be re-applied with different parameters or another reasoner can be used. As a result, prototype reasoners can be added without jeopardizing the entire decomposition process. The variety of decomposition reasoners available is more important, as some reasoners might define hex-meshing strategies resulting in poor element quality, and regions that are not covered by any reasoner will either need manual decomposition or receive a tet mesh. The current limitation comes from the ability to apply virtual topology split operations, as the workflow requires the application of many operations successively and any failed operation will make any subsequent split invalid.

Beyond facilitating the integration of the different tools, having a pre-processing workflow based on virtual topology makes this approach compatible with traditional applications of virtual topology for de-featuring and geometry clean-up. The update of the decomposition is also made simpler. Sub-regions can be recombined by the virtual topology merge operator without rolling back the entire decomposition (e.g. the decomposition in Figure 12 (a) can be obtained from the one in Figure 12 (b)). The constraints associated with the meshing strategies can also be used to automatically propagate CAD design updates to the decomposition [30].

One can argue that using virtual geometry curves and extracting explicit regions for some reasoners is incompatible with the notion of virtual topology. Even simple geometries can result in complex block decompositions (see Figure 14 (a)) and inferring cutting geometry solely from the topological

requirements would create skewed angles and potentially inverted geometry that are easily avoided using virtual geometry curves. Extracting temporary geometric regions is the only realistic way currently available for a user to interact with the analysis topology and allows integration of a wide range of off-the-shelf tools within a given CAD environment.



**Figure 14. (a) Fully blocked model and (b) mesh file generated directly from CAD using virtual entities.**

The analysis topology is essential to maintain the meshing strategies at interfaces in manifold environment and ensure that the final decomposition is suitable for meshing. Meshing strategies translate downstream meshing constraints into constraints on the decomposition that are available from within a CAD environment. The process of propagating imprints and decomposing source faces of sweepable regions may create more subset regions than necessary for achieving a good quality hex mesh, but this results in sub-regions that are simpler to mesh and compatible with a wider range of meshing tools. Eventually, a model that has been fully decomposed into block regions is in itself a very coarse hex mesh. It can be refined and meshed directly from the virtual decomposition in CAD, without having to commit the geometric decomposition and transfer to a CAE environment. In Figure 14 (b), all edges are discretized as per the meshing recipe and nodal positions on surface and inside the volume are identified using transfinite interpolation [31], before being written to a Nastran deck input file to define a mesh. On the other hand, sweepable regions with paved source faces can accommodate pair of singularities that cancel each other, redirecting the flow of the elements. This offers more freedom for node location and avoids the propagation of small element size from small details to the entire mesh.

Automatically propagating imprints is also beneficial for semi-automated decompositions workflows, as automatic partition of neighbor regions reduces the amount of work for the operator. Manual intervention can also unlock regions that are suitable for automatic processing, hence the impact of user input is maximized and no time is wasted carrying out repetitive decomposition tasks.

## 8. CONCLUSION

A method to integrate various automated decomposition reasoners in a single incremental decomposition workflow has been presented. All the split operations are applied using

virtual topology to build an analysis topology that stores and maintains interface information and meshing strategies. This analysis topology along with virtual geometry curves are used to abstract the actual analysis model, enabling reasoners and manual users to operate on a model that is equivalent to the analysis model before it is created. Each reasoner identifies and extracts sweepable and block regions, and the meshing strategies associated with each region are used to propagate splits across interfaces to ensure everything remains hex-meshable. With all the necessary information for mesh automation available, the CAD model is decomposed to create an analysis model that can be exported to a meshing tool.

## 9. FUTURE WORK

Future research directions include:

- Extending the range of decomposition reasoners, in particular frame-field based methods that also focus on the handling of singularity lines.
- Integrating automatic de-featuring reasoners to remove small fillets and holes using virtual topology.
- Further investigating virtual topology meshing capabilities, including sub-mapping and paving.

## ACKNOWLEDGMENTS

The authors wish to acknowledge the financial support provided by Innovate UK through the COLIBRI (ref 113296) project. We also thank Rolls-Royce for permission to publish this paper.

## REFERENCES

- [1] J. Sarrate, E. Ruiz-Gironés, and X. Roca, "Unstructured and Semi-Structured Hexahedral Mesh Generation Methods," *Comput. Technol. Rev.*, vol. 10, pp. 35–64, 2014.
- [2] S. J. Owen *et al.*, "An Immersive Topology Environment for Meshing," in *Proceedings of the 16th International Meshing Roundtable*, Berlin, Heidelberg: Springer, 2008, pp. 553–577.
- [3] Y. Lu, R. Gadh, and T. J. Tautges, "Feature based hex meshing methodology: feature recognition and volume decomposition," *Comput. Des.*, vol. 33, no. 3, pp. 221–232, Mar. 2001, doi: 10.1016/S0010-4485(00)00122-6.
- [4] D. White, L. Mingwu, and S. Benzley, "Automated hexahedral mesh generation by virtual decomposition," in *Proceedings of the 4th International Meshing Roundtable*, 1995, pp. 165–176.
- [5] L. Sun, C. M. Tierney, C. G. Armstrong, and T. T. Robinson, "An enhanced approach to automatic decomposition of thin-walled components for hexahedral-dominant meshing," *Eng. Comput.*, vol. 34, no. 3, pp. 431–447, Nov. 2018, doi: 10.1007/s00366-017-0550-x.



- [6] H. Wu, S. Gao, R. Wang, and J. Chen, "Fuzzy clustering based pseudo-swept volume decomposition for hexahedral meshing," *Comput. Des.*, vol. 96, pp. 42–58, Mar. 2018, doi: 10.1016/J.CAD.2017.10.001.
- [7] X. Roca and J. Sarrate, "Local dual contributions: Representing dual surfaces for block meshing," *Int. J. Numer. Methods Eng.*, vol. 83, pp. 709–740, 2010, doi: 10.1002/nme.2852.
- [8] N. Kowalski, F. Ledoux, M. L. Staten, and S. J. Owen, "Fun sheet matching: towards automatic block decomposition for hexahedral meshes," *Eng. Comput.*, vol. 28, no. 3, pp. 241–253, Jul. 2012, doi: 10.1007/s00366-010-0207-5.
- [9] R. Wang, C. Shen, J. Chen, H. Wu, and S. Gao, "Sheet operation based block decomposition of solid models for hex meshing," *Comput. Des.*, vol. 85, pp. 123–137, Apr. 2017, doi: 10.1016/J.CAD.2016.07.016.
- [10] M. A. Price and C. G. Armstrong, "Hexahedral Mesh Generation by Medial Surface Subdivision: Part II. Solids with Flat and Concave Edges," *Int. J. Numer. Methods Eng.*, vol. 40, no. 1, pp. 111–136, 1997.
- [11] "Sandia National Laboratories: index," <https://cubit.sandia.gov/> (accessed Aug. 03, 2021).
- [12] J. H.-C. Lu, W. R. Quadros, and K. Shimada, "Evaluation of user-guided semi-automatic decomposition tool for hexahedral mesh generation," *J. Comput. Des. Eng.*, vol. 4, no. 4, pp. 330–338, Oct. 2017, doi: 10.1016/J.JCDE.2017.05.001.
- [13] J. H.-C. Lu, I. Song, W. R. Quadros, and K. Shimada, "Volumetric Decomposition via Medial Object and Pen-Based User Interface for Hexahedral Mesh Generation," *Proc. 20th Int. Meshing Roundtable, IMR 2011*, pp. 179–196, 2011, doi: 10.1007/978-3-642-24734-7\_10.
- [14] T. Blacker, "Automated Conformal Hexahedral Meshing Constraints, Challenges and Opportunities," *Eng. Comput.*, vol. 17, no. 3, pp. 201–210, Oct. 2001, doi: 10.1007/PL00013384.
- [15] K. Miyoshi and T. Blacker, "Hexahedral Mesh Generation Using Multi-Axis Cooper Algorithm Cubit Mesh Generation," in *Proceedings of the 9th International Meshing Roundtable*, 2000, pp. 89–97.
- [16] H. Wu, S. Gao, R. Wang, and M. Ding, "A global approach to multi-axis swept mesh generation," *Procedia Eng.*, vol. 203, pp. 414–426, Jan. 2017, doi: 10.1016/J.PROENG.2017.09.817.
- [17] A. Sheffer, M. Bercovier, T. Blacker, and J. Clemets, "Virtual Topology Operators for Meshing," *Int. J. Comput. Geom. Appl.*, vol. 10, no. 03, pp. 309–331, Jun. 2000, doi: 10.1142/s0218195900000188.
- [18] C. M. Tierney, L. Sun, T. T. Robinson, and C. G. Armstrong, "Using virtual topology operations to generate analysis topology," *Comput. Des.*, vol. 85, pp. 154–167, 2017, doi: 10.1016/j.cad.2016.07.015.
- [19] K. Ho-Le, "Finite element mesh generation methods: a review and classification," *Comput. Des.*, vol. 20, no. 1, pp. 27–38, 1988, doi: 10.1016/0010-4485(88)90138-8.
- [20] T. D. Blacker and M. B. Stephenson, "Paving: A new approach to automated quadrilateral mesh generation," *Int. J. Numer. Methods Eng.*, vol. 32, no. 4, pp. 811–847, 1991, doi: 10.1002/nme.1620320410.
- [21] C. M. Tierney *et al.*, "Efficient Symmetry-Based Decomposition for Meshing Quasi-Axisymmetric Assemblies," *Comput. Des. Appl.*, vol. 16, no. 3, pp. 478–495, 2019, doi: 10.14733/cadaps.2019.478-495.
- [22] L. Sun, C. M. Tierney, C. G. Armstrong, and T. T. Robinson, "Decomposing complex thin-walled CAD models for hexahedral-dominant meshing," *Comput. Aided Des.*, vol. 103, pp. 118–131, Dec. 2018, doi: 10.1016/j.cad.2017.11.004.
- [23] B. Lecallard *et al.*, "Automatic Hexahedral-Dominant Meshing for Decomposed Geometries of Complex Components," *Comput. Des. Appl.*, vol. 16, no. 5, pp. 846–863, 2019, doi: 10.14733/cadaps.2019.846-863.
- [24] N. J. Taylor and R. Haimes, "Geometry modelling: Underlying concepts and requirements for computational simulation (invited)," *2018 Fluid Dyn. Conf.*, 2018, doi: 10.2514/6.2018-3402.
- [25] J. Kripac, "A mechanism for persistently naming topological entities in history-based parametric solid models," *Comput. Des.*, vol. 29, no. 2, pp. 113–122, Feb. 1997, doi: 10.1016/S0010-4485(96)00040-1.
- [26] B. Lecallard, "Virtual topology based hex-dominant meshing and re-meshing," PhD thesis, Queen's University Belfast, 2020.
- [27] "NX | Siemens Digital Industries Software," <https://www.plm.automation.siemens.com/global/fr/products/nx/> (accessed Aug. 03, 2021).
- [28] T. S. Li, C. G. Armstrong, and R. M. McKeag, "Quad mesh generation for k-sided faces and hex mesh generation for trivalent polyhedra," *Finite Elem. Anal. Des.*, vol. 26, no. 4, pp. 279–301, Aug. 1997, doi: 10.1016/S0168-874X(96)00085-6.
- [29] "ITI - International TechneGroup | CADfix," <https://www.iti-global.com/cadfix> (accessed Mar. 06, 2019).
- [30] B. Lecallard, C. M. Tierney, T. T. Robinson, C. G. Armstrong, D. C. Nolan, and A. E. Sansom, "Updating and Re-meshing Virtually Decomposed Models," in *Proceedings of the 28th International Meshing Roundtable*, 2019, pp. 50–67.
- [31] L. E. Eriksson, "Generation of Boundary-Conforming Grids Around Wing-Body Configurations Using Transfinite Interpolation," *Aiaa J.*, vol. 20, no. 10, pp. 1313–1320, 1982, doi: 10.2514/3.7980.

# INTERACTIVE VISUALIZATION OF LARGE AND ARBITRARY POLYGONAL AND POLYHEDRAL MESHES WITH OPENGL 4

Matthieu Maunoury<sup>1</sup>

Rémi Feuillet<sup>2</sup>

Adrien Loseille<sup>3</sup>

*Inria Saclay, Gamma Team, 1 Rue Honoré d'Estienne d'Orves, 91120 Palaiseau, France.  
{matthieu.maunoury<sup>1</sup>, remi.feuillet<sup>2</sup>, adrien.loseille<sup>3</sup>}@inria.fr*

## ABSTRACT

This paper describes an efficient strategy to visualize polygons and polyhedra using OpenGL 4 flexibility. Such meshes offer flexibility as the number of vertices and faces are arbitrary. Dual meshes are examples of polygonal and polyhedral meshes. We give explanations on how polygons and polyhedra can efficiently be stored in mesh files. Algorithms to tessellate polygons into triangles are described. Many examples and comparisons with another visualization software show that our methodology is efficient (about 40 times faster than ParaView). Interactivity is also ensured with post-processing tools such as picking and cut planes.

**Keywords:** Visualization, Polygonal Meshes, Polyhedral Meshes, OpenGL 4, GLSL, Shaders

## 1. INTRODUCTION

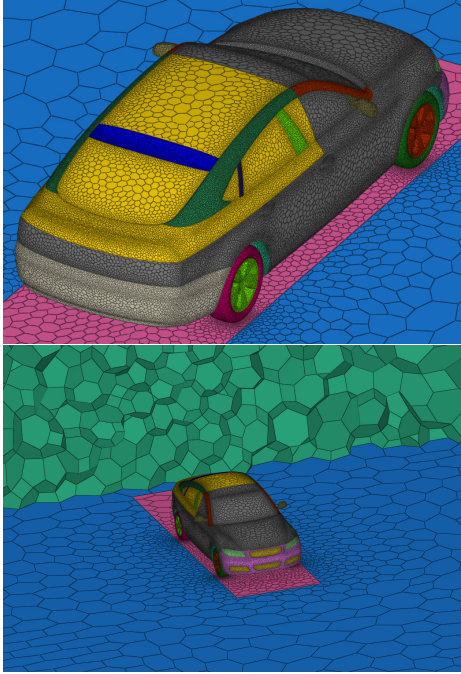
The design of efficient meshing techniques or the developments of new numerical schemes requires the ability to quickly load, visualize and inspect meshes and solutions. The efficiency is bounded by what we can see and should be possible on classic laptops and workstations. This process becomes critical when non linear elements are used. This is the case for polygonal meshes where a few effective rendering techniques exist.

The goal of numerical simulations is to predict the behavior of physical phenomenons without using prototypes or experimentations. Many domains are involved such as Computational Fluid Dynamic, acoustics, electromagnetism, or biomedical. In general, the numerical simulations pipeline is composed of a mesh generation step [1], then a problem is numerically solved with the help of this mesh and finally a numerical solution is obtained. All along the process, visualization is needed to check and validate the mesh and the solution, and give tools to analyze the results. The choice of the elements types in the mesh depends

on the type of equations studied or on the solver. The most common elements are triangles and quadrilaterals for surfaces and tetrahedra and hexahedra for volumes but prisms or pyramids are sometimes used, especially when hybrid meshes are involved. Unlike the latter elements, one interest to use polygonal (for surfaces) and polyhedral (for volumes) meshes is the flexibility as elements have an arbitrary number of vertices. Figure 1 shows examples of such meshes.

Only a few commercial meshers and simulation packages such as **Simcenter StarCCM+** [2] or **OpenFOAM** [3] handle generic polygons and polyhedra. There has been little works on generation of polygonal and polyhedral meshes [4, 5, 6], some of them are generated as the dual of tetrahedral meshes. Other works focus on the construction of finite element interpolants on polygonal and polyhedral meshes [7, 8, 9, 10, 11, 12].

However, many visualization software programs do not handle polygons and when it is the case, the interactivity is often limited. It is also the case when high-order elements and solutions are considered. There are two main strategies: ray-casting with possibly



**Figure 1:** Examples of meshes composed of polygons only (top) and polygons and polyhedra (bottom). The number of vertices is not constant by element.

volume visualization and low-order remeshing. The first approach is ray-casting with volume visualization [13, 14]. A significant limitation of this technique is the cost and as a consequence it does not compete with the interactivity of the standard linear rendering methods. Furthermore, in the case of polygons and polyhedra, the cell-to-cell connectivity is required to traverse the mesh and needs lots of memory that is a limiting factor. The second approach is the low-order remeshing: the idea is to tessellate each element into triangles for surfaces or tetrahedra for volumes. Then, any visualization software is able to render these elements. For instance, **ParaView** [15] or **VisIt** [16] use this technique to represent polygons and polyhedra. New mesh visualization software solutions have also emerged [17, 18, 19, 20, 21, 22].

The goal of this paper is to explain how polygonal and polyhedral meshes are visualized using OpenGL 4. For this purpose, many points are detailed such as: storage and I/O, how tessellation of polygons into triangles is done, how post-processing tools like picking, clip planes is done. The tessellation algorithms presented in this paper do not add extra vertices in the tessellation as the aim is to minimize the number of triangles to maximize the rendering performances. All works presented in this paper have been developed in **ViZiR 4** that is freely available in its dedicated web

site <http://vizir.inria.fr>.

The paper is outlined as follows. Section 2 is devoted to storage and I/O of polygons and polyhedra. Section 3 gives a presentation of the OpenGL 4 graphic pipeline. Section 4 tackles the problem of tessellation of polygons. Section 5 deals with post-processing tools and interactivity. Examples are given all along the paper and more complex examples are described in Section 6. Comparisons are done in this paper between **ParaView**, **VisIt** and the current approach.

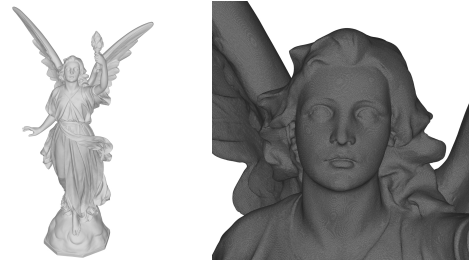
## 2. STORAGE AND I/O OF POLYGONS AND POLYHEDRA

All the results collected in this paper have been generated with the same laptop: a MacBook Pro with details given in Table 1.

Hardware	Details
CPU	Intel Core i7 2.6 GHz 6-core
GPU	AMD Radeon Pro Vega 20 4 Gb
Mem	32 Gb of RAM 2400 MHz DDR4
OS	Mac

**Table 1:** Hardware used for testing.

A key to have an efficient visualization is to be able to quickly open mesh and solution files. Input and output are handled by the **libMeshb**<sup>1</sup> library. The files follow the **GMF** format provided by this library. For instance, the mesh of Lucy (see Fig. 2) with more than 14 millions vertices and 28 millions triangles (642 Mb) is opened in less than 1.5 seconds.



**Figure 2:** Rendering of a large mesh of 14M vertices and 28M triangles in 7.5 seconds (total time) on a laptop.

One difficulty to define polygons and polyhedra is that the number of vertices may be different for each element. It means that the number of vertices and number of faces for polyhedra must be defined for each element. In the following, some vocabulary is given and the storage of polygons and polyhedra is explained.

<sup>1</sup><https://github.com/LoicMarechal/libMeshb>

- Boundary polygons: polygons that are displayed. Each element is characterized by an arbitrary number of vertices and the indices of these vertices. In practice, a list of all boundary polygons vertices is defined and for each element the beginning index and a reference are given. The number of vertices is deduced by looking at the index of beginning of the next element (i.e. the ending index is the one prior the starting index of the next element). It allows to access any element independently of all the previous polygons and very quickly.
- Inner polygons: polygons that are not displayed but are useful to define polyhedra: these polygons are faces of polyhedra. The definition of these inner polygons is the same as for boundary polygons: the beginning index of each inner polygon and the list of indices of these inner polygons vertices are given.
- Polyhedra: polyhedra that are displayed when intersecting the clip plane. In practice, a list of all polyhedra's faces (i.e. inner polygons) is defined and for each element the beginning index and a reference are given. Following the same idea than for polygons, the number of faces is deduced by looking at the index of beginning of the next element (i.e. the ending index is the one preceding the beginning index of the next element) and each element is accessed very quickly. Note that in practice, the faces of volume elements are rendering.

New keywords have been introduced to the `libMeshb` library to define these polygons and polyhedra. Furthermore, some functions have been introduced to ease the access of these data. All vertices are stored only once and are used to define boundary polygons, inner polygons and polyhedra. Then, all these elements are stored separately. This way, only boundary polygons are taken into account to display surfaces whereas inner polygons and polyhedra are only considered for cut plane. Information are generally stored in binary format to be more efficient but can also be written in ASCII format. The format follows the `libMeshb` library.

### 3. PRESENTATION OF THE OPENGL 4 GRAPHIC PIPELINE

The OpenGL 4 rendering pipeline can be customized with up to five different shader stages (see Fig. 3). These shaders are GLSL source code files that replace parts of the OpenGL pipeline. In general, a shader receives its input via developer-defined input variables, and the data for those variables come either from the

main OpenGL application or previous pipeline stages (other stages). Data can also be provided to any shader using uniform variables or textures [23]. More details on OpenGL 4 and in particular OpenGL Shading Language (GLSL) can be found in [23, 24].

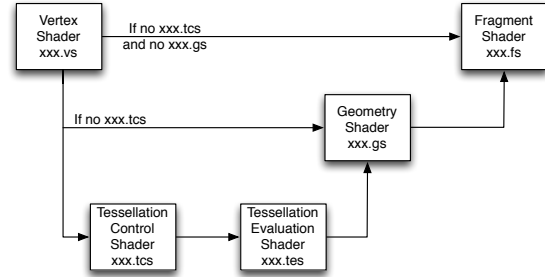


Figure 3: Shaders used for the OpenGL graphic pipeline.

Two shaders are enough to define a graphic pipeline, the vertex shader and the fragment shader. The vertex shader handles the vertices. The data corresponding to the vertices positions are transformed into clip coordinates. The fragment shader determines the color for each pixel. Many parameters affect the color like a shading, a solution, an isoline, or a wireframe rendering. For the storage of raw data (like high-order solutions), textures are used.

Besides these two shaders, a geometry shader can be added to govern the processing of primitives. It allows to create new geometries on the fly. With this in mind, it can be preceded by the two tessellation shaders: the tessellation control shader and the tessellation evaluation shader. They are used to control the tessellation of the primitives, in other words, in how many sub-elements the elements should be divided.

OpenGL 4 graphic pipeline flexibility allows to compute on the fly the solution. It leads to a pixel exact rendering when flat elements (of degree one) are considered regardless of the degree of the solution. This recent language (GLSL) enables `ViZiR 4` to certify a faithful and interactive depiction. High order solutions are natively handled by `ViZiR 4` on surface and volume (tetrahedra, pyramids, prisms, hexahedra) meshes which can naturally be hybrid.

When more complex geometries are considered, curved elements perform a better approximation of the geometry. In this case, tessellation shaders occur in OpenGL pipeline (see [20, 21] for more details on the shaders pipeline) to tessellate all elements directly on the GPU. For solutions on such curved elements, almost pixel exact rendering is ensured [21].

## 4. TESSELLATION OF POLYGONS INTO TRIANGLES

### 4.1 Why tessellate polygons into triangles

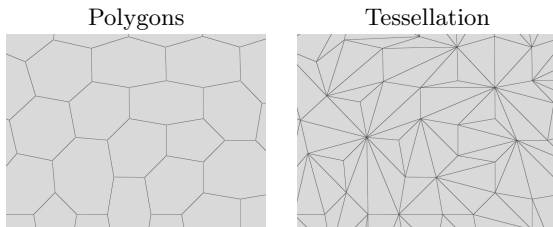
As explained in Section 3, only the vertex and the fragment shaders are mandatory. For instance, in the case of triangles (of degree 1), the geometry shader could be useful for example to compute and display normals vectors but in many cases, this shader is avoided. The reason is that the use of this shader is expensive. To illustrate this cost, a comparison is done on the mesh of Lucy (28M triangles, see Fig. 2) and is outlined in Table 2. In the first case, only the vertex and fragment shaders are used, and the number of Frames Per Second is 28. In the second case, a geometry shader is added, and does nothing more than pass data through itself (i.e. the rendering is exactly the same than in the first case) and the number of Frames Per Second falls to 6. The rendering of triangles has good performance because only vertex and fragment shaders are used while additional shaders are necessary for more complex elements. For this reason, polygons are tessellated into triangles. However, when High Order elements are displayed, Geometry and Tessellation shaders should be used as it is the only way to have a good rendering (done on the GPU) of them.

GLSL pipeline	FPS
Vertex + Fragment Shaders	28
Vertex + Geometry + Fragment Shaders	6

**Table 2:** Comparisons on Lucy mesh (28M triangles) of FPS when Geometry shader is used or not.

### 4.2 How to tessellate polygons

We consider only simple polygons, which means polygons with no two non-consecutive edges intersecting, these polygons are convex or concave. An example of tessellation used for the rendering is shown in Fig. 4.



**Figure 4:** Rendering of polygons (left) and their tessellation into triangles (right) used for the rendering.

#### 4.2.1 Edge visibility

During the creation of the tessellation, the visibility (i.e. a Boolean) of the 3 edges of each triangle is defined. Indeed, some edges need to be visible as their correspond to the boundary of a polygon while others should not be visible as lying inside the polygon. Textures are used to send the information on visibility (booleans) to the fragment shader so that the appropriate color can be set according to the position (inside or on the boundary of the polygon) of the edge.

#### 4.2.2 Definitions of normal by polygon

Normals of elements are important because they are used in the shading, for instance in Phong model [25]. Usually, the normal is computed for each element and given to the shaders by textures. To have a smoother shading, one normal  $n_{poly}$  for each polygon is defined. Let's first define:

$$\mathcal{A} = \sum_{i=1}^{d-2} (P_{i+2} - P_1) \wedge (P_{i+1} - P_1) \quad (1)$$

where  $d$  is the number of vertices of the polygon,  $P_i$  the points of the polygon and  $\wedge$  the usual vector cross product. Note that this definition of normal depends on the choice of the first vertex. Actually, the choice of the first vertex is not important as it can be modified, the crucial thing is to keep this definition during the whole process. Finally, the normal of polygon  $n_{poly}$  is obtained after normalization:

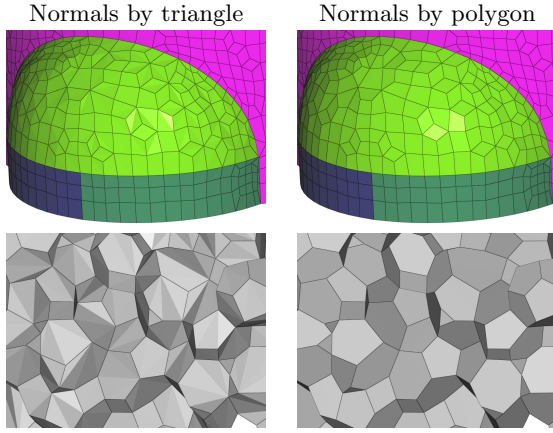
$$n_{poly} = \frac{\mathcal{A}}{\|\mathcal{A}\|} \quad (2)$$

Fig. 5 shows a comparison of these two types of normals (by triangle and by polygon) and the definition of normal by polygon given by (2) gives a better smoothness of the shading.

#### 4.2.3 A first naive tessellation algorithm

A first naive tessellation algorithm is to create all triangles from one vertex, for instance the first one. The number of created triangles is  $d - 2$  where  $d$  is the number of vertices of the polygon. This algorithm is described in Algorithm 1. Fig. 6 shows an example of use of Algorithm 1 for a very simple 5-sides convex polygon.

Note that if the polygon is a triangle (3-sides), all the edges are set to visible. Now, let us consider another 5-sides polygon but which is concave. To do so, the second point of Fig. 6 is simply moved to become a concave point as shown in Fig. 7. Algorithm 1 is used,



**Figure 5:** Comparison of normals by triangle (left) and by polygon (right) with a Phong model as shading.

---

**Algorithm 1:** A first naive tessellation algorithm

---

**Input:**  $d$ ,  $\text{pol}$  (list of vertices of size  $d$ ).

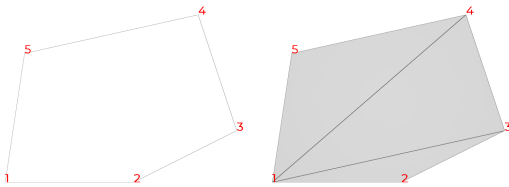
**Output:**  $\text{NmbTri}$  (number of triangles),  $\text{Tri}$  (list of indices of triangles),  $\text{VisEdg}$  (edges visibility)

```

(1)  $\text{NmbTri} = d - 2$  ;
   if  $\text{NmbTri} < 1$  then
       return 0 ;
   for  $i = 1$  to  $\text{NmbTri}$  do
(2)    $\text{Tri}[i][1] = \text{pol}[1]$  ;
(3)    $\text{Tri}[i][2] = \text{pol}[i + 1]$  ;
(4)    $\text{Tri}[i][3] = \text{pol}[i + 2]$  ;
(5)    $\text{VisEdg}[i][1] = 1$  ;
(6)    $\text{VisEdg}[i][2] = 0$  ;
(7)    $\text{VisEdg}[i][3] = 0$  ;
   end
(8)  $\text{VisEdg}[1][3] = 1$  ;           //– First triangle
(9)  $\text{VisEdg}[\text{NmbTri}][2] = 1$  ;   //– Last triangle
   //– If only 1 triangle, all edges are visible
   if  $\text{NmbTri} == 1$  then
(10) |  $\text{VisEdg}[0][3] = 1$  ;

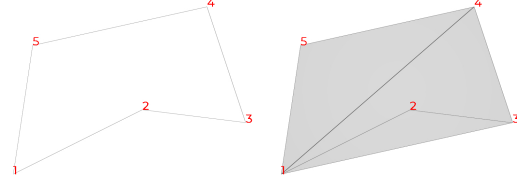
```

---



**Figure 6:** A 5-sides convex polygon. The naive tessellation algorithm 1 works.

and the same tessellation is constructed but this time it fails as the polygon is concave. The tessellation does not span the polygon as the triangle  $\{1, 2, 3\}$  is outside the polygon.

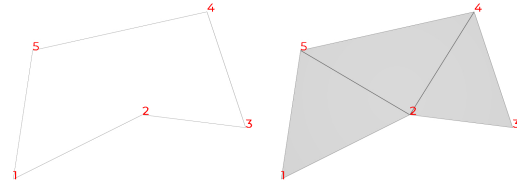


**Figure 7:** A 5-sides concave polygon. The naive tessellation algorithm 1 fails.

Fortunately, there is a very simple criterion to know if a triangle is outside the polygon. Once the tessellation has been created, it is sufficient to check if all dot products between the normal of the polygon, defined by (2), and the normals of triangles are positive. If all these dot products are positive, no triangle will lie outside the polygon and algorithm 1 is applied. For the first case (Fig. 6), this criterion is true while it is false for the second case (Fig. 7) as the dot product between the normal of the polygon and the normal of triangle  $\{1, 2, 3\}$  is negative. Thus, algorithm 1 can not be applied and a more general tessellation algorithm is needed.

#### 4.2.4 Choosing a better a starting point in the naive tessellation algorithm

Algorithm 2 sums up the process to check if algorithm 1 should be used. If only one concave vertex has been found, the idea is to generate the tessellation from this point: Algorithm 1 is then used with this point as a starting point. An example is shown in Fig. 8 with the same concave polygon than for Fig. 7 but this time the tessellation is correct. Note that the criterion of dot products positiveness is also checked for every created triangles. Indeed, this new tessellation could be incorrect and in this case, a more general algorithm described in Section 4.2.5 should be used.



**Figure 8:** A 5-sides concave polygon. Following algorithm 2, a concave vertex (here vertex 2) has been used as a starting point to create the tessellation.

---

**Algorithm 2:** Check if the first naive tessellation algorithm 1 can be used

---

**Input:**  $d$ ,  $\text{pol}$  (list of vertices of size  $d$ ).  
 Compute the normal polygon  $n_{poly}$  with eq. (2) ;  
 Create the tessellation following algorithm 1 ;  
 $\text{NmbVerConcave} = 0$  ;  
**for**  $i = 1$  **to**  $\text{NmbTri}$  **do**  
   Compute normal  $n_{tri}$  of triangle  $\text{Tri}[i]$  ;  
   **if**  $n_{tri} \cdot n_{poly} < 0$  **then**  
     Algorithm 1 can not be used. ;  
     Update the list of concave vertices ;  
      $\text{NmbVerConcave}++$  ;  
**end**  
**if**  $\text{NmbVerConcave} = 0$  **then**  
   Tessellation created with algo. 1 is correct.  
**else if**  $\text{NmbVerConcave} = 1$  **then**  
   Launch algorithm 1 with the concave point as  
   a starting point to create new tessellation  
   and check it with algorithm 2. ;  
**else**  
   Too much concave points, another algorithm  
   is needed (see algorithm 3). ;  
**end**

---

## 4.2.5 A general tessellation algorithm

Triangulating a polygon, that is decomposing a polygon into a set of triangles is a problem that have been investigated for a long time. One of the most famous method is the ear clipping theorem [26, 27, 28, 29]. The principle is that in each polygon, an ear can be found and removed from the polygon. The result is a polygon whose area is smaller. By doing this recursively, a set of triangles is obtained and span all the polygon. Note that most of ear-clipping algorithms are for 2D (polygons in a plane) only whereas we are considering 3D meshes.

---

**Algorithm 3:** General tessellation algorithm

---

**Input:**  $d$ ,  $\text{pol}$  (list of vertices of size  $d$ ).

- (1) Compute the normal polygon  $n_{poly}$  with eq. (2) ;
  - (2) Project all polygon vertices into an orthogonal plane of the normal polygon ;
  - (3) **while**  $d > 3$  **do**
  - (4)   Find a triangle which is admissible. ;
  - (5)   Update all the lists: add this triangle to the tessellation, remove this triangle from the polygon list ;
  - end**
  - (6) Generate the last triangle with the 3 last points. ;
- 

Algorithm 3 gives the general steps to create a tessellation following the idea of the ear clipping theorem. All details of this algorithm are described now:

**(1) Normals computations.** The normal of the polygon is defined following eq. (2).

**(2) Points projections.** All vertices of the polygon are projected on a same plane that is orthogonal to the normal  $n_{poly}$  of the polygon. These projected points  $\mathcal{P}_i$  are obtained following:

$$\mathcal{P}_i = P_i - (P_i, n_{poly}) n_{poly} \quad (3)$$

where  $P_i$  denotes the vertex  $i$  of the polygon and  $(.,.)$  is the usual dot product.

**(3) Find an ear.** Once a triangle has been found, the polygon changes, its size becomes smaller as one vertex is removed from the list of the polygon.

**(4) Find an admissible triangle.** To find a triangle that is correct, the idea is to take three consecutive vertices of the polygon and check if the dot product between the normal polygon and the normal of the triangle is positive and that no other projected point of the polygon lies inside this projected triangle. To do so, let's note  $\mathcal{P}_{i_1}$ ,  $\mathcal{P}_{i_2}$  and  $\mathcal{P}_{i_3}$  the three consecutive projected points and  $\mathcal{P}_j$  another projected point of the polygon, we define

$$\begin{aligned} u &= (\mathcal{P}_j \mathcal{P}_{i_1} \wedge \mathcal{P}_j \mathcal{P}_{i_2}) \cdot n_{poly} \\ v &= (\mathcal{P}_j \mathcal{P}_{i_2} \wedge \mathcal{P}_j \mathcal{P}_{i_3}) \cdot n_{poly} \\ w &= (\mathcal{P}_j \mathcal{P}_{i_3} \wedge \mathcal{P}_j \mathcal{P}_{i_1}) \cdot n_{poly} \end{aligned} \quad (4)$$

where  $(. \wedge .)$  denotes the usual cross product. Then, if  $u$ ,  $v$  and  $w$  are all positive, the point is inside the triangle. If none of the other projected points  $\mathcal{P}_j$  lies inside the triangle  $\{\mathcal{P}_{i_1} \mathcal{P}_{i_2} \mathcal{P}_{i_3}\}$ , this triangle is admissible.

**(5) Updates.** All the lists must be updated. One more triangle is added in the tessellation,  $\text{Tri}$  is updated with these three vertices. The visibility of edges  $\text{VisEdg}$  is set by looking at the local index of the vertices. Indeed, if the two points of the edge are consecutive (or are the first and last vertices of the original polygon),  $\text{VisEdg}$  is set to 1, otherwise it is set to 0. Then, the vertex  $\mathcal{P}_{i_2}$  is removed from the list  $\text{pol}$  of the polygon and  $d$  is decreased by one unit.

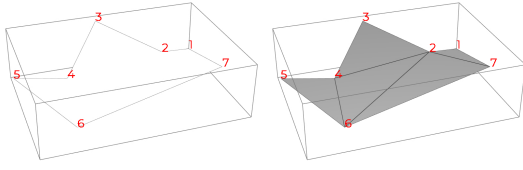
**(6) Last triangle.** Finally, a last triangle is created with the last three vertices.  $\text{Tri}$  and  $\text{VisEdg}$  are updated with the same way than (5).

Figure 9 shows an example of a 7-sides concave polygon handled with algorithm 3.

If the element is ill-defined, for instance not a simple polygon by with intersected edges, it is possible that neither algorithm 2 nor algorithm 3 work. In this case, algorithm 1 can still be applied to generate a tessellation in order to at least be able to see the polygon.

To sum up the tessellation process, algorithm 1 is first used. If the tessellation is correct according positive-ness criterion, there is no need to use the other algorithms. If there is only one concave vertex, this point is



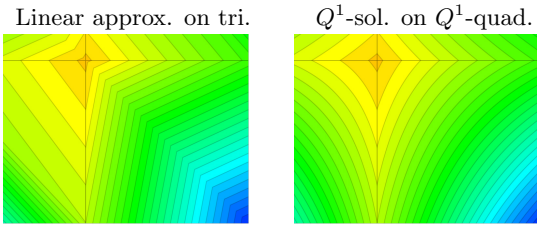


**Figure 9:** A 3-dimensional 7-sides concave polygon. With algorithm 3, even a polygon with two concave vertices is handled.

used as a starting point in algorithm 1 and we check if the tessellation is correct. Otherwise, the most general tessellation algorithm, algorithm 3 is launched to create the tessellation. All these tessellation algorithms are done in the CPU before the rendering of these triangles by the GPU.

### 4.3 Solution rendering with a tessellation of triangles

If a solution has been computed on polygons, a representation from a tessellation of triangles might be inaccurate. For instance, let's consider a mesh of quadrilaterals and a solution defined at vertices. Thus, the solution is  $Q^1$ -solution on a  $Q^1$ -quadrilaterals and is therefore bi-linear as shown in Fig. 10. If a tessellation of two triangles is generated with affine functions on them is done, an approximation is created and the representation is inaccurate (the tessellation in 2 triangles can be guessed in Fig. 10). Note that the rendering in both cases is pixel-exact and isolines are displayed to highlight the linearity or non-linearity of the solution plotted.



**Figure 10:** Rendering of  $Q^1$ -solution on  $Q^1$ -quadrilaterals (right) and tessellation into triangles with affine representation (left).

## 5. POST-PROCESSING TOOLS AND INTERACTIVITY

Many post-processing tools are available to make the analysis of results possible. Some of them are pre-

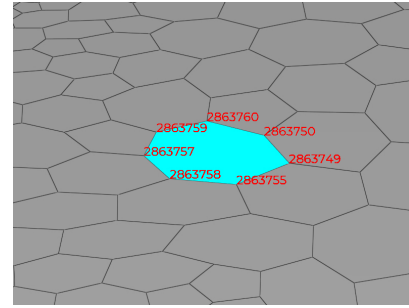
sented in this section. Such an interactivity is fundamental to develop and validate new algorithms.

### 5.1 Picking and hiding surfaces by reference

Any element can be picked to get information. When an element is picked, it is colored in light blue and the number of its vertices appear in red as shown in Fig. 11. More information is printed on the terminal, for example the element picked in Fig. 11:

```
Polygon (7-sides)      19793 : [ 2863749
2863755 2863758 2863757 2863759
2863760 2863750] Ref 3
```

The printed information: the number of vertices (sides), the index of element, the indices of all vertices and the reference number of the polygon.

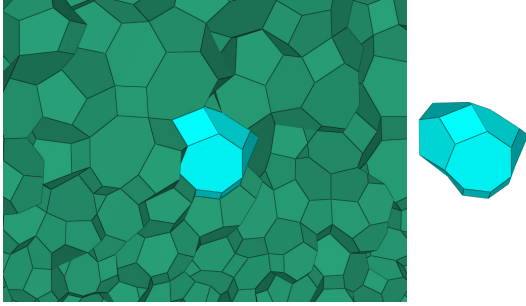


**Figure 11:** Picking a polygon (in light blue).

When a face of a volume element (here a polyhedron) is picked, all the faces of this volume elements are also set in light blue as shown in Fig. 12. In the same way that for polygons, information are printed on the terminal: the number of faces (inner polygons), the index of the polyhedron, the list of indices of these faces and the polyhedron reference. Then, for each face, the number of vertices and the list of vertices are printed. Here is an example for an hexahedron:

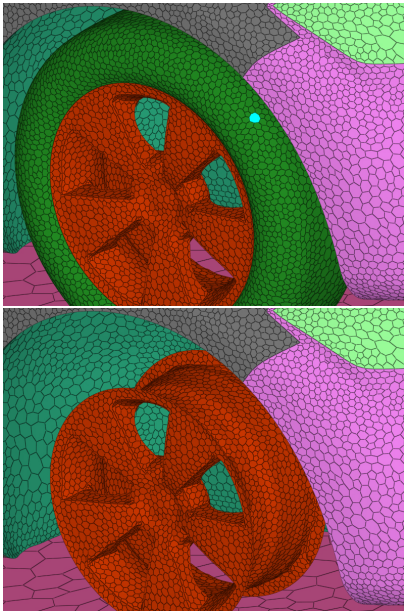
```
Polyhedron (6-faces)
952607 : [385682647 385683053 385685045
385685051 385685052 385685053] Ref 0
Face (4-sides)      1 : [ 69594081
69594078 69581014 69581016]
Face (4-sides)      2 : [ 69594081
69559331 69559333 69594078]
Face (4-sides)      3 : [ 69559331
69594081 69581016 69559334]
Face (4-sides)      4 : [ 69559333
69594078 69581014 69559341]
Face (4-sides)      5 : [ 69559331
69559333 69559341 69559334]
Face (4-sides)      6 : [ 69581016
69559334 69559341 69581014]
```

To inspect meshes, it is interesting to hide some elements. After an element is picked, it is possible to



**Figure 12:** Picking a polyhedron (in light blue). Left: with all elements in the cut plane. Right: the polyhedron alone.

hide all elements having the same reference id (corresponding typically to a patch or a specific part of the object). An example is shown in Fig. 13 where the green surface (tire) is hidden to show the elements behind.



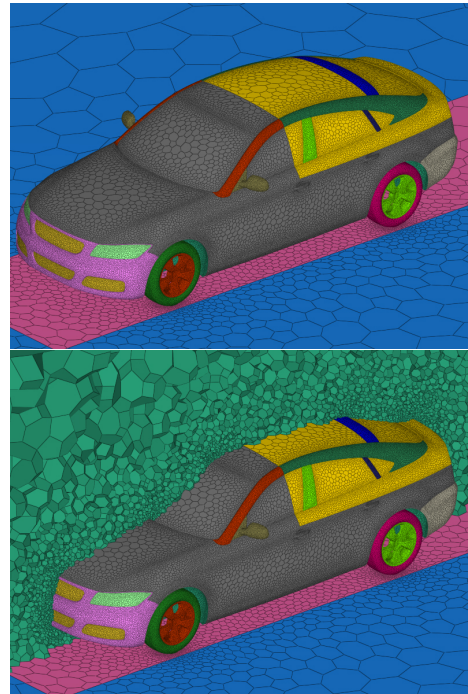
**Figure 13:** Example of picking (first picture) and hiding by reference (second picture).

In practice, here are the steps to set the whole element picked in light blue. A triangle has been picked. The index of the polygon or polyhedron is known (previously stored). All triangles belonging to the same element (polygon or polyhedron) are added to the picked list. During the creation of the texture that gives the rgba (red, green, blue and alpha) of the triangle to the shaders, if the triangle is in the picked list, its rgb is set to (0, 1, 1), that is light blue, instead of the color that

should be displayed (for example its reference color or the usual signature grey). In the fragment shader, if the color texture is (0, 1, 1), that is light blue, we know that the triangle has been picked. If it is a polygon (i.e. not a volume element), `Fragcolor` is set to (0, 255, 255, 1), so that it will appear in light blue independently of the shading. Otherwise, the color is light blue but the shading can be seen as in Fig. 12.

## 5.2 Clip planes

To visualize polyhedra, clip planes are used. The clip plane can be defined by its equation. Otherwise, the clip plane can be translated or rotated with the mouse from an initial state. Then, all polyhedra belonging to this cut plane are displayed. In practice, the faces of these volume elements are rendered. To find if a polyhedron is intersected by the cut plane, one just have to look at the sign of all the element vertices in the cut plane equation  $ax + by + cz + d$ , where  $x$ ,  $y$  and  $z$  are the coordinates of the vertex and  $a$ ,  $b$ ,  $c$  and  $d$  the parameters of the cut plane equation [20, 21]. If some of them are positive and some others are negative, the volume element lies in the cut plane. Fig. 14 shows an example of clip plane.



**Figure 14:** Examples of rendering without (first picture) and with (second picture) clip plane.

## 6. EXAMPLES AND COMPARISONS

### 6.1 Comparisons with other visualization software

Some comparisons are made with **ParaView** and **VisIt**. Several meshes of different sizes are studied where the geometry is the car plotted in Fig. 14. The **VTK Unstructured Grid (vtu)** format is used in **ParaView** and in **VisIt**. Meshes were converted from **CGNS** (CFD General Notation System) to **vtu** format. The versions 5.7 of **ParaView** and 3.2.1 of **VisIt** are used. For **ViZiR 4**, meshes were converted from **CGNS** to **libMeshb** format. Table 3 compares the total rendering time that is the time to open the mesh file, add objects to the scene and render the mesh. Three cases, similar than Fig. 14, are taken into account. The number of boundary polygons are 439 170, 1 101 804 and 2 649 542 and the number of triangles created by the tessellation are respectively 1 705 918, 4 333 706 and 10 505 154 in **ViZiR 4**. It gives an average of 3.88, 3.93 and 3.96 triangles per polygons and show that the number of vertices is truly arbitrary with an average of 6 for each boundary polygon. The ratio are huge and is explained by the fact that the time to open the mesh file in **ParaView** and in **VisIt** is long and because a surface reconstruction is done in **ParaView** and in **VisIt** and this step is very expensive. In **ViZiR 4**, the surface reconstruction is not done (even if it could be called) as this information (boundary polygons) is already in the mesh file. Otherwise, the surface reconstruction can be done in a pre-processing step, and then should be saved in the mesh. Note that this step can still be done in the visualization software but is useless and time-consuming if it has already be stored in the mesh file.

	Case 1	Case 2	Case 3
# vertices	9 600 780	24 551 880	61 321 116
# polygons	439 170	1 101 804	2 649 542
# polyhedra	2 652 618	6 603 843	15 055 285
<b>ViZiR 4</b> (s)	1.93	4.48	10.98
<b>ParaView</b> (s)	81.7	204.0	505.8
Ratio / <b>ParaView</b>	42.3	45.5	46.1
<b>VisIt</b> (s)	86.9	219.3	582.8
Ratio / <b>VisIt</b>	45.0	48.9	53.1

**Table 3:** Comparison of total rendering wall time (s) including mesh files opening.

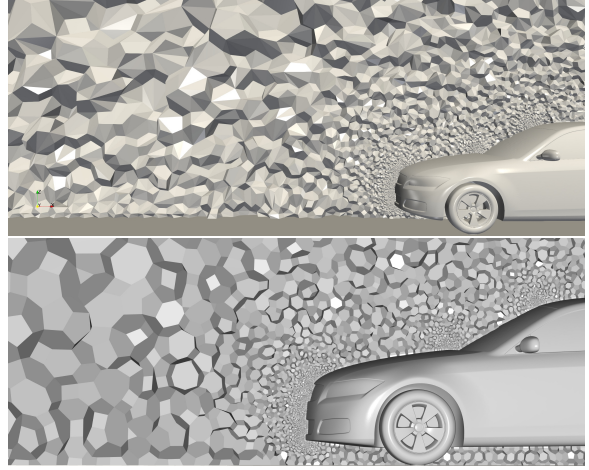
Table 4 compares the time to generate cut planes. In **ParaView**, these cut planes are crinkle clips. **VisIt** was not able to generate crinkle clips with these meshes. Again, the ratio are huge and mainly due to the slow surface reconstruction done in **ParaView**.

Fig. 15 shows a comparison of rendering obtained from

	Case 1	Case 2	Case 3
# vertices	9 600 780	24 551 880	61 321 116
# polygons	439 170	1 101 804	2 649 542
# polyhedra	2 652 618	6 603 843	15 055 285
<b>ViZiR 4</b> (s)	0.6	1.4	3.1
<b>ParaView</b> (s)	57.9	147.0	357.9
Ratio	98.1	106.5	114.0

**Table 4:** Comparison of wall time (s) to generate cut planes (clip).

**ParaView** and **ViZiR 4**. It is clear that the shading in **ParaView** is done by triangles while it is done by polygon in **ViZiR 4** as explained in Section 4.2.2 and gives a better smoothness. Note that with **VisIt**, instead of the polygons, the triangles (i.e the tessellation) are displayed.



**Figure 15:** Comparison of rendering obtained from **ParaView** (first picture) and **ViZiR 4** (second picture). In the first case, shading is done by triangle while it is done by polygon in the second case.

Additional metrics, mesh file size, memory and video memory used, and the number of frames per second have been sum up in Table 5 for the 3 same cases than the previous tables. For **ViZiR 4**, the format **meshb** (binary) from **libMeshb** is used while the **vtu** format is used in **ParaView**. The mesh size in **cgns** format has also been added. Both software programs have good FPS for all cases and therefore interactive enough. **ParaView** needs much more memory especially during the preparation of the rendering however requires much less video memory.

Finally, last comparisons have been done with another CPU-GPU combo: Windows 11-Nvidia. The laptop is a 6-core i7 3 Ghz with 32 GB of RAM and the graphic card a Nvidia quadro T1000 (4Gb). Results are very



	Case 1	Case 2	Case 3
Size .meshb (us)	753 M	1.9 G	4.6 G
.vtu ParaView	1.8 G	4.5 G	11 G
.cgns ParaView	1.0 G	2.6 G	6.4 G
FPS (us)	60	60	52
FPS ParaView	58	57	46
RAM (us)	1.59 G	3.91 G	9.47 G
RAM ParaView	2.14 G	4.98 G	12.03 G
Peak RAM ParaView	2.95 G	7.46 G	17.59 G
VRAM (us)	214 M	453 M	1.01 G
VRAM ParaView	182 M	250 M	435 M

**Table 5:** Comparisons of additional metrics: mesh sizes, FPS (frames per second), memory RAM and VRAM (video RAM) for ParaView and ViZiR 4.

similar than Table 3 which is not surprising as the two laptops have similar features. For the first case, ViZiR 4 has a total rendering wall time of 1.68 s and 1.37 Gb RAM while ParaView needs 50 s (ratio 30) and 2.4 Gb of RAM. For the second case, ViZiR 4 has a total rendering wall time of 4.92 s and 4.67 Gb RAM while ParaView needs 2 min 17 (ratio 32) and 5.3 Gb of RAM. In all cases, both programs have very good frame rates.

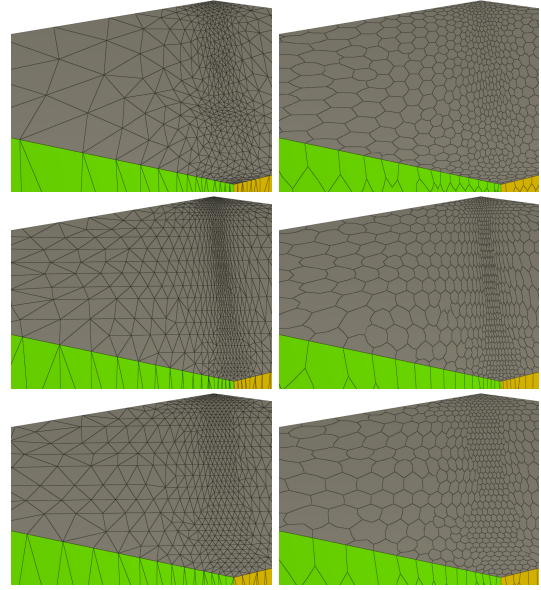
## 6.2 Examples

Dual meshes are implicit supports for many CFD solvers, for instance the ones based on finite volume methods. Here we illustrate with Fig. 16, 17 and 18 some rendering to study the differences between adaptive meshing techniques. Examples of Fig. 18 come from the same airplane geometry than Fig. 17 with a zoom on the wings. Classic mesh adaptation techniques are based on a sequence of local mesh modifications. These techniques consist in an advancing-point methods using metric fields. We can see that the dual patterns are different.

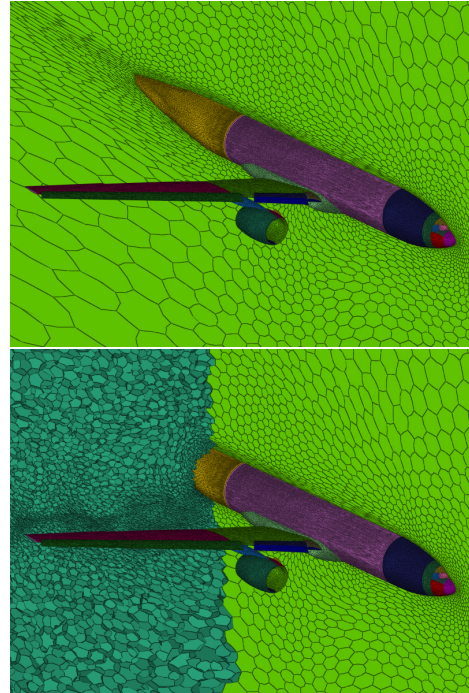
## 7. CONCLUSIONS

In this paper, we presented how OpenGL 4 can be used to visualize polygonal and polyhedral meshes. In particular, we discussed how the storage in the mesh file is done. We showed that the use of triangles in the OpenGL graphic pipeline is the most efficient as it is the simplest. For this reason, polygons are tessellated into triangles. Algorithms and criteria are given to create a good tessellation. Many examples show the efficiency of our method and comparisons with ParaView and VisIt show that ViZiR 4 is much faster.

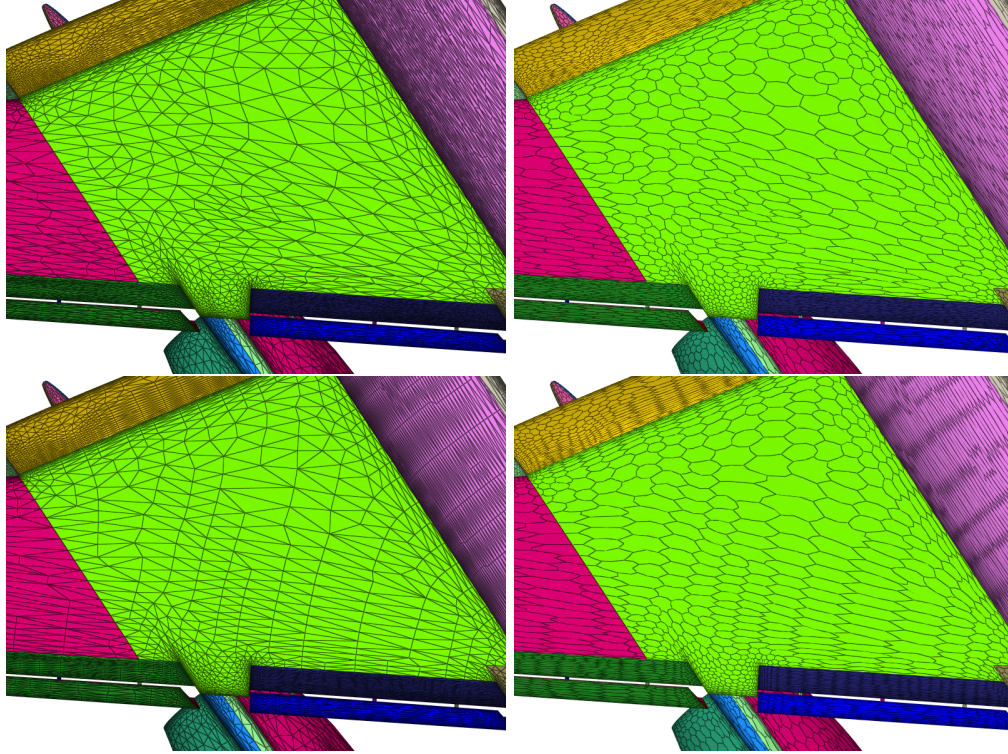
A first perspective of this work could be to use polygons when capping is done. Indeed, when cut planes are generated, two modes can be used: cut plane (or



**Figure 16:** Primal (left) and dual (right) meshes for standard adaptation (first line), and two metric-aligned techniques (second and third lines).



**Figure 17:** An example of polyhedral mesh without (first picture) and with (second picture) clip plane.



**Figure 18:** Primal (left) and dual (right) meshes for standard adaptation (top) and metric-aligned adaptation (bottom).

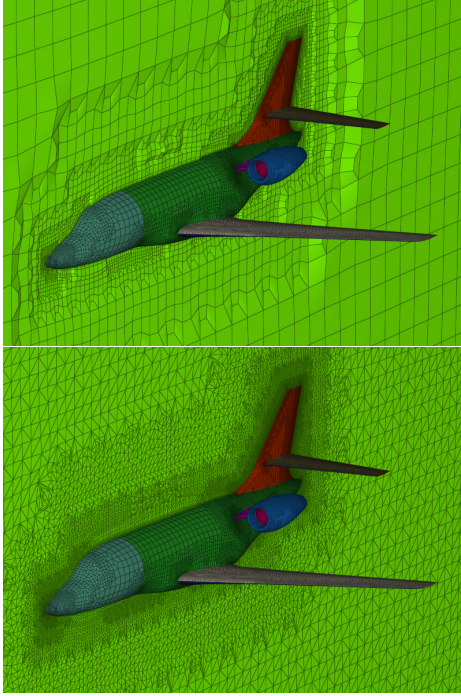
crinkle) when the faces of the volume elements are shown or capping (or slice), when the intersection of the volume element with the plane is computed. In the latter case, when non simplicial elements such as prisms, pyramids or hexahedra are considered, the intersection is in fact a polygon. At the moment, triangles are used to display the capped elements as shown in Fig. 19 where the mesh is composed of quadrilaterals and hexahedra. Another perspective would be to handle solutions on polygonal and polyhedral meshes. Finally, as OpenGL 4 is able to handle high-order elements, one can imagine that high-order polygons, when and if it will exist, could also be visualized with OpenGL 4.

## 8. ACKNOWLEDGMENTS

This work was supported by the public grant ANR Impacts, reference ANR-18-CE46-0003. The authors are also grateful to Loïc Maréchal (Inria) for providing the `libMeshb` library and his help in testing, Lucien Rochery (Inria) for fruitful discussions on tessellation algorithms and Siemens for providing meshes from `Simcenter StarCCM+`.

## References

- [1] Frey P.J., George P.L. *Mesh generation: application to finite elements*. Iste, 2007
- [2] “Simcenter STAR-CCM+.” <https://www.plm.automation.siemens.com/global/en/products/simcenter/STAR-CCM.html>
- [3] “OpenFOAM.” <https://www.openfoam.com>
- [4] Oaks W., Paoletti S. “Polyhedral mesh generation.” *Proceedings of the 9th International Meshing Roundtable*, pp. 57–67. 2000
- [5] Paoletti S. “Polyhedral mesh optimization using the interpolation tensor.” *Proceedings of the 11th International Meshing Roundtable*, pp. 19–28. 2002
- [6] Garimella R.V., Kim J., Berndt M. “Polyhedral mesh generation and optimization for non-manifold domains.” *Proceedings of the 22nd International Meshing Roundtable*, pp. 313–330. Springer, 2014
- [7] Wachspress E.L., EL W. “A rational finite element basis.” 1975



**Figure 19:** Example of clip plane (first picture) and capping (second picture) of a hexahedral mesh.

- [8] Sukumar N. “Construction of polygonal interpolants: a maximum entropy approach.” *International journal for numerical methods in engineering*, vol. 61, no. 12, 2159–2181, 2004
- [9] Sukumar N., Malsch E. “Recent advances in the construction of polygonal finite element interpolants.” *Archives of Computational Methods in Engineering*, vol. 13, no. 1, 129, 2006
- [10] Cangiani A., Georgoulis E.H., Houston P. “hp-version discontinuous Galerkin methods on polygonal and polyhedral meshes.” *Mathematical Models and Methods in Applied Sciences*, vol. 24, no. 10, 2009–2041, 2014
- [11] Manzini G., Russo A., Sukumar N. “New perspectives on polygonal and polyhedral finite element methods.” *Mathematical Models and Methods in Applied Sciences*, vol. 24, no. 08, 1665–1699, 2014
- [12] Perumal L. “A brief review on polygonal/polyhedral finite element methods.” *Mathematical Problems in Engineering*, vol. 2018, 2018
- [13] Muigg P., Hadwiger M., Doleisch H., Hauser H. “Scalable hybrid unstructured and structured grid raycasting.” *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, 1592–1599, 2007
- [14] Muigg P., Hadwiger M., Doleisch H., Groller E. “Interactive volume visualization of general polyhedral grids.” *IEEE transactions on visualization and computer graphics*, vol. 17, no. 12, 2115–2124, 2011
- [15] KitWare Inc. “ParaView.” <https://www.paraview.org/>
- [16] Childs H., Brugger E., Whitlock B., Meredith J., Ahern S., Pugmire D., Biagas K., Miller M., Harrison C., Weber G.H., Krishnan H., Fogal T., Sanderson A., Garth C., Bethel E.W., Camp D., Rübel O., Durant M., Favre J.M., Navrátil P. “VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data.” *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pp. 357–372. Oct 2012
- [17] Musy M., Dalmasso G., Sullivan B. “marcomusy/vtkplotter: vtkplotter.”, 2019
- [18] Sullivan B., Kaszynski A. “PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK).” *Journal of Open Source Software*, vol. 4, no. 37, 1450, 2019
- [19] Canepa A., Infante G., Hitschfeld N., Lobos C. “Camarón: An Open-source Visualization Tool for the Quality Inspection of Polygonal and Polyhedral Meshes.” *International Conference on Computer Graphics Theory and Applications*, vol. 2, pp. 130–137. SCITEPRESS, 2016
- [20] Loseille A., Feuillet R. “Vizir: High-order mesh and solution visualization using OpenGL 4.0 graphic pipeline.” *56th AIAA Aerospace Sciences Meeting, AIAA Scitech*, 2018
- [21] Feuillet R., Maunoury M., Loseille A. “On pixel-exact rendering for high-order mesh and solution.” *Journal of Computational Physics*, vol. 424, 109860, 2021
- [22] Bracci M., Tarini M., Pietroni N., Livesu M., Cignoni P. “HexaLab. net: An online viewer for hexahedral meshes.” *Computer-Aided Design*, vol. 110, 24–36, 2019
- [23] Wolff D. *OpenGL 4.0 Shading Language Cookbook*. Packt Publishing, 2011
- [24] Sellers G., Wright R., Haemel N. *OpenGL Super-Bible, Sixth Edition*. Addison-Wiley, 2013
- [25] Phong B.T. “Illumination for computer generated pictures.” *Communications of the ACM*, vol. 18, no. 6, 311–317, 1975



- [26] Meisters G.H. “Polygons have ears.” *The American Mathematical Monthly*, vol. 82, no. 6, 648–651, 1975
- [27] Tarjan R.E., Van Wyk C.J. “An  $O(n \log \log n)$ -time algorithm for triangulating a simple polygon.” *SIAM Journal on Computing*, vol. 17, no. 1, 143–178, 1988
- [28] Chazelle B. “Triangulating a simple polygon in linear time.” *Discrete & Computational Geometry*, vol. 6, no. 3, 485–524, 1991
- [29] ElGindy H., Everett H., Toussaint G. “Slicing an ear using prune-and-search.” *Pattern Recognition Letters*, vol. 14, no. 9, 719–722, 1993



# $P^3$ BÉZIER CAD SURROGATES FOR ANISOTROPIC MESH ADAPTATION

A. Loseille<sup>1</sup>

L. Rochery<sup>2</sup>

<sup>1</sup>*Inria Saclay, 91120 Palaiseau, France, adrien.loseille@inria.fr*

<sup>2</sup>*Inria Saclay, 91120 Palaiseau, France, lucien.rochery@inria.fr*

## ABSTRACT

Mesh generation and adaptation rely heavily on BREPs created by proprietary CAD software, piecewise parametric descriptions of geometry from which numerous problems arise: model continuity is only enforced up to a tolerance often higher than required mesh sizes, projection is costly and prone to error, derivatives — thus normals and curvature metrics — may not be well defined, unintended small features driving unnecessary mesh complexity may be present... unlike discrete surface meshes, of which high-order ones offer advantageous convergence speed over degree of freedom ratio relative to  $P^1$  meshes.  $P^3$  meshes, in particular, are the first degree for which  $\mathcal{G}^1$  continuity at the vertices may be enforced. In this paper, we compare two methods to construct  $P^3$  meshes from a CAD model. The resulting  $P^3$  meshes are then used instead of the CAD model in a full converging adaptation loop on a complex geometry, the HL-CRM wing with flaps with a highly anisotropic metric field.

**Keywords:** BREP, CAD surrogate,  $P^3$  Bézier triangles, anisotropic mesh adaptation, surface mesh generation

## 1. INTRODUCTION

Cost-effective numerical resolution of PDEs — namely of hyperbolic ones such as Navier-Stokes — is enabled by anisotropic mesh adaptation. Using either generic [1, 2] or PDE-tailored [3] error estimates, meshes are locally modified [4, 5] or the degree of interpolation is locally elevated (p-adaptation) [6, 7], sometimes both (hp-adaptation) [8, 9], to match local features of the solution and thus maximize the precision over degrees of freedom (computational cost) ratio [10, 11]. This places mesh generation and adaptation at the heart of simulation, which becomes a loop that converges to an optimal mesh-solution couple (Fig. 1).

Domain geometry is described in a continuous fashion using a CAD file. The BREP (Boundary REPresentation) model with rational Bézier patches and curves [12, 13], in particular, is widely used and the focus of this paper. In this framework, a surface is described as a collection of connected trimmed patches. Global

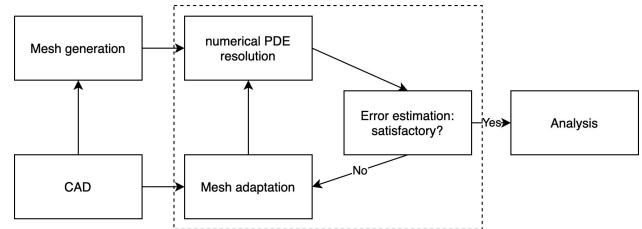
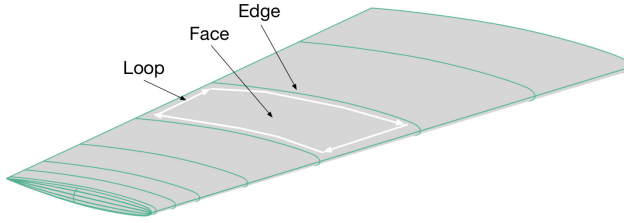


Figure 1: Mesh adaptation loop

BREP topology is illustrated in Fig. 2. This description using Bézier and rational Bézier curves and surfaces is quite flexible and can represent a wide variety of shapes with complex features, as well as represent exactly a number of frequently used geometric primitives such as sections of spheres, cylinders, cones... Some of the more frequent operations on these objects include evaluating a point on the surface given its parametric coordinates, projection of 3D points

onto CAD edges and faces and computing derivatives (up to the second order, most frequently) at given points on the surface. These operations are intensively used both in initial mesh generation as well as in subsequent adaptation steps, as illustrated in Fig. 1. BREP models are a tool specialized in defining shapes rather than manipulating them. For this reason, it is not infrequent for CAD models to pose a number of difficulties, such as by not being watertight, having face lines that degenerate into single points, autointersecting faces, interpenetrating neighbouring faces, ill-conditioned parameterizations, unintended small features, etc... [14, 15] go over the reasons for these features in detail. Furthermore, these errors are often bound by tolerances too high for the purposes of mesh adaptation where smaller edges may be required close to or on the surface [16].



**Figure 2:** Trimmed BREP topology

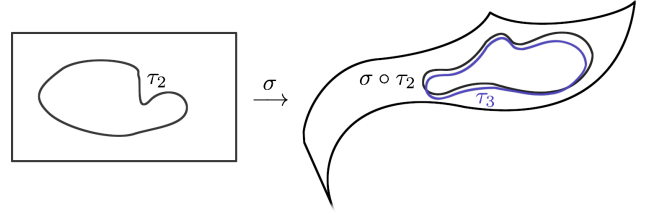
Strategies to combat these issues operate at essentially two levels: by correcting the CAD before it is used in meshing, and by devising robust tessellation algorithms. By tessellation, we designate any surface mesh which will not necessarily be used as the support for any volume mesh, but rather as a discrete surrogate for the CAD geometry. Strategies of the first type include the use of virtual topology [17] and various CAD correction procedures [15].

In this paper, we present our approach of the second type, and illustrate the ability of  $P^3$  surface meshes to drive mesh adaptation as well as the CAD itself on cases with high anisotropy close to the boundary. A  $P^1$  tessellation of the parametric surface is first generated, and then elevated to the third order. A discrete tessellation offers fast inverse evaluation through simple algorithms, it can be seen as a first order Taylor expansion of the surface. Therefore, projection on a  $P^1$  mesh is similar to gradient descent on the original surface with derivatives precomputed (Jacobians of surface elements). In fewer words, it is much simpler and stable, much of the burden being shifted to the tessellation step. This is only possible at the expense of precision, a cost that can be reduced by replacing linear elements by higher-order elements such as  $P^3$  Bézier triangles. Through simple benchmarks, the speed and precision of projection on these meshes are exhibited. They are then used as the geometric support for a full

adaptation loop using a highly anisotropic analytical boundary layer metric.

## 1.1 The BREP

Let us go over, in little detail, the elements that constitute a BREP-based CAD model. Further details can be found in the reference [13]. A model is given by a set of faces, loops, edges and corners. CAD edges are mapped from 1D domains and grouped into loops. These loops trim CAD faces which are mapped from 2D domains. In turn, faces sharing a loop portion are neighbours. Edges are defined using a 1D domain  $D_1 \subset \mathbb{R}$  and a parameterization  $\tau_3 : D_1 \mapsto \mathbb{R}^3$ . The parameterization  $\tau_3$  is, typically, a rational Bézier curve although it is not infrequent for CAD systems to distinguish subcases such as particular conics and circle arcs. Faces are defined much in the same way, from a 2D domain  $D_2 \subset \mathbb{R}^2$  and a parameterization  $\sigma : D_2 \mapsto \mathbb{R}^3$ . Likewise, it is typical for  $\sigma$  to be a rational Bézier surface, but particular cases such as sections of spheres and cylinders are sometimes distinguished. The face trimming is then defined using another parameterized curve defined on  $D'_1 \subset \mathbb{R}$  with a mapping  $\tau_2 : D'_1 \mapsto D_2$  that draws a 2D curve in the face's parameter domain.



**Figure 3:** Trimming curve definition tolerances:  $\tau_2$  defined in surface patch parameter space is mapped to the black curve in  $\mathbb{R}^3$  by  $\sigma$ ,  $\tau_3$  maps a segment to the blue curve. These curves differ by a tolerance set by the CAD design tool (visible here as the exaggerated gap).

This curve is a representation in face parameter space of the physical face-face intersection curve. Unfortunately, one cannot assume that  $\sigma \circ \tau_2 = \tau_3$ . This is due to the fact that these trimming curves are computed from rational Bézier surface intersections which, in the general case, are not rational Bézier curves. This means that  $\tau_3$  is, typically, already an approximation of the desired curve. On top of this,  $\tau_2$  is, again, an approximation of the projection of  $\tau_3$  onto the face parameter space  $D_2$ . As such, CAD software only guarantees correspondence between the face-local and global curves up to a tolerance, typically much higher than desired mesh sizes. This means that despite the fact that parametric surfaces and curves are arbitrarily precise, they can be very inaccurate in some regions (junctions), where the geometry is actually ill-defined

under a threshold. Concretely, this means that there can actually be gaps between faces meant to share an edge, or that neighbouring faces may be interpenetrating at scales that cannot be neglected by the meshing algorithm. The paper [16] goes over this issue in great detail, illustrating the impact on mesh adaptation for CFD problems. For instance, it is frequent for wing and fuselage intersections to be given with a tolerance several orders of magnitude higher than the smallest prescribed mesh size in this area by the end of adaptation. Furthermore, derivatives of the parameterizations are used to compute metric fields for surface error approximation [18, 19] or surface normals and tangent planes. Once more, the CAD description may pose problems such as by having faces map portions of edges onto points (degeneracy) or having unwanted local features such as folds under the tolerance (another issue pointed out in [16]). In particular, derivatives are not defined at these points and unstable in their vicinity. This also means that these situations call for robust optimization algorithms when projecting points close to these regions and that projection is slow (relative to on a discrete mesh) and prone to error.

## 2. THE PARAMETRIC $P^3$ MESHER

In this section we turn to the parametric meshing algorithm. It takes a CAD object as input and outputs a  $P^1$  mesh adapted with regards to a geometric approximation metric. The  $P^3$  mesh is then constructed by elevating the degree of these  $P^1$  elements and projecting control nodes.

### 2.1 Building the initial tessellation

In this section, we give a rough description of a parametric surface mesher. Let us consider a trimmed CAD face  $F$ . It is defined by a rectangular parametric domain  $D \subset \mathbb{R}^2$ , a rational Bézier function  $\sigma : D \mapsto \mathbb{R}^3$ , and a set of connected rational Bézier edges  $E_i \subset \mathbb{R}^3$  forming a closed loop. These edges  $E_i$  are constructed by the CAD system on path intersections and are an approximation of the actual intersection. Indeed, two rational surfaces need not intersect at a rational curve. The projection of these edges on  $D$  is also given by the CAD system as a two-dimensional rational Bézier curve lying in  $D$ . If we denote  $\tau_i : [0, 1] \mapsto \mathbb{R}^3$  the parametrization of  $E_i$  in physical space and  $\tau_i^{(2)} : [0, 1] \mapsto D$  the parametrization of the given projection of  $E_i$  onto  $D$ , the identity

$$\sigma_i \circ \tau_i^{(2)} = \tau_i$$

does not hold in the general case.

The first step of our method is to produce a  $P^1$  mesh of the trimming loop  $\bigcup E_i$  in physical space. This step proceeds on a per-edge basis and computes an

edge approximation error metric [18] to construct a mesh with quasi-uniform geometric error under the prescribed tolerance (user input). This mesh gives a set of vertices  $P_i$  and edges  $e_i = [P_i, P_{i+1}]$  in  $\mathbb{R}^3$ . These vertices are then projected onto  $D$ , giving  $P_i^{(2)}$  s.t.  $P_i = \sigma(P_i^{(2)})$ . The edges between these projected points form a closed loop in  $D$ . We use this as the trimming curve instead of  $\tau_i^{(2)}$ . In doing so we guarantee that, if a given CAD edge is part of the trimming loops of two edge faces, it will be mapped by both parametrizations to the same physical line mesh.

A constrained Delaunay mesher in parametric domain  $D$  is then called with this boundary as input. This first tessellation is used to compute the surface approximation error metric in low [18] or in high order [19]. When constructing the  $P^1$  mesh as support for the  $P^3$  surface mesh, a  $P^3$  geometric approximation metric is used to adapt the surface mesh. The trimmed patch tessellation can then be adapted to this metric field [20, 21].

### 2.2 The $P^3$ surface mesh

$P^3$  Bézier elements are defined by a set of control nodes, either the Lagrange nodes which we denote  $P_{ijk}^\ell$  or the Bézier nodes  $P_{ijk}$ , with  $(i, j, k) \in \hat{K}^d = \{(i, j, k) \in \mathbb{N}^3, i + j + k = 3\}$ . In the latter case, the mapping from the reference triangle  $\hat{K}$  is given by

$$F_K(\xi) = \sum_{\alpha \in \hat{K}^d} B_\alpha(\xi) P_\alpha,$$

whereas in the former,  $F_K$  is interpolated exactly by the degree three Lagrange basis  $(\phi_\alpha)_{\alpha \in \hat{K}^d}$ ,

$$F_K(\xi) = \sum_{\alpha \in \hat{K}^d} \phi_\alpha(\xi) F_K(\hat{P}_\alpha) = \sum_{\alpha \in \hat{K}^d} \phi_\alpha(\xi) P_\alpha^\ell$$

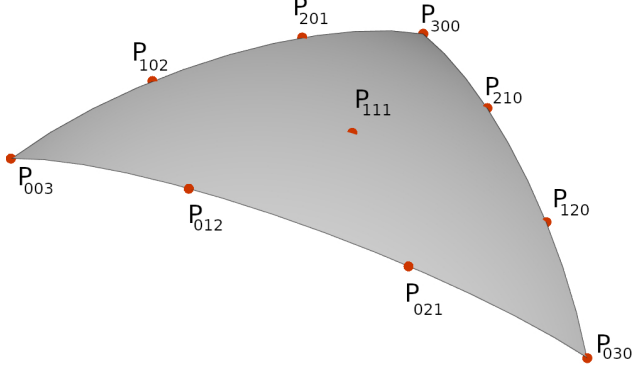
where the  $\hat{P}_\alpha$  are the control nodes of the reference element, *i.e.*  $\hat{P}_{ijk} = (i/3, j/3, k/3)$ . This also provides the definition of the Lagrange control nodes. The  $B_\alpha$  are the Bernstein polynomials, defined by

$$B_{i,j,k}(u, v, w) = \binom{3}{i} \binom{3-i}{j} u^i v^j w^k$$

for every  $(i, j, k) \in \hat{K}^d$  and  $(u, v, w) \in \hat{K}$ . When choosing the control points at the thirds, *i.e.*

$$P_{ijk} = \frac{i}{3} P_{300} + \frac{j}{3} P_{030} + \frac{k}{3} P_{003},$$

the Lagrange and Bézier control nodes coincide and the mapping  $F_K$  degenerates to become linear. This is what we'll naturally refer to as the straight  $P^3$  element. Figure 4 illustrates a  $P^3$  triangle with its Lagrange control nodes.



**Figure 4:**  $P^3$  triangle with Lagrange nodes in red.

The  $P^1$  mesh must now be brought to the third degree. To do this, new edge and face high-order nodes must be created for each triangle and have their positions set. There are two approaches to this. The first is to initialize the Lagrange nodes of each element at the straight position in physical space. These points are then projected onto the surface using the CAD model. The second approach evaluates the Lagrange nodes on CAD faces directly. This can be done easily since, for every vertex  $P$  of the  $P^1$  mesh, its coordinates in the parametric domain of the host face are known from the  $P^1$  meshing step. Taking an edge of extremities  $P_{30}$ ,  $P_{03}$  on a face with mapping  $\sigma$  and with  $\xi_{30}$  and  $\xi_{03}$  known such that  $\sigma(\xi_i) = P_i$ , the Lagrange nodes for the direct approach are given by

$$P_{21}^{dir} = \sigma\left(\frac{2}{3}\xi_{30} + \frac{1}{3}\xi_{03}\right) \text{ and } P_{12}^{dir} = \sigma\left(\frac{1}{3}\xi_{30} + \frac{2}{3}\xi_{03}\right),$$

whereas, denoting by  $\Pi_\sigma$  the surface projection operator, the Lagrange nodes for the inverse approach are

$$P_{21}^{proj} = \Pi_\sigma\left(\frac{2}{3}P_{30} + \frac{1}{3}P_{03}\right), \quad P_{12}^{proj} = \Pi_\sigma\left(\frac{1}{3}P_{30} + \frac{2}{3}P_{03}\right).$$

The direct approach is faster than the inverse approach due to CAD projections. However, we will see that the indirect approach is more robust, since the initial position of the Lagrange nodes is not too far from the straight element, which is valid in the sense that it does not self-intersect and remains well-conditioned for optimization (projection). In either case, boundary edges are first brought to the high order and stored in a hash table. Triangles are then looped over and new control points created or recovered from the hash table. At this stage, the mesh is fully  $P^3$  with all control points on the geometry. The last optional step is to apply a Lagrange-to-Bézier transformation since the Bézier representation is a more convenient and generalizable one. Using the definition of the Lagrange control nodes, we have the following relations:

$$\sum B_{ijk}^3(i_0/3, j_0/3, k_0/3) P_{ijw} = P_{i_0j_0k_0}^\ell$$

for every  $(i_0, j_0, k_0) \in \widehat{K}^d$ . These equations degenerate for any triplet where one of the indices is 3 (principal vertices of the triangle), leaving 7 non-trivial equations which correspond to two per edge and one for the face control node. Equations relating to edge control nodes are treated in pairs, yielding 6 of the Bézier control nodes. The face Bézier node is finally computed using these values. Taking as example the edge  $\{w = 0\}$ ,

$$\begin{aligned} 12P_{210} + 6P_{120} &= 27P_{210}^\ell - 8P_{300} - P_{030} \\ 6P_{210} + 12P_{120} &= 27P_{120}^\ell - 8P_{030} - P_{300}. \end{aligned}$$

This leads to

$$\begin{aligned} -6P_{210} &= 9P_{120}^\ell - 18P_{210}^\ell - 2P_{030} + 5P_{300} \\ -6P_{120} &= 9P_{210}^\ell - 18P_{120}^\ell - 2P_{300} + 5P_{030}. \end{aligned}$$

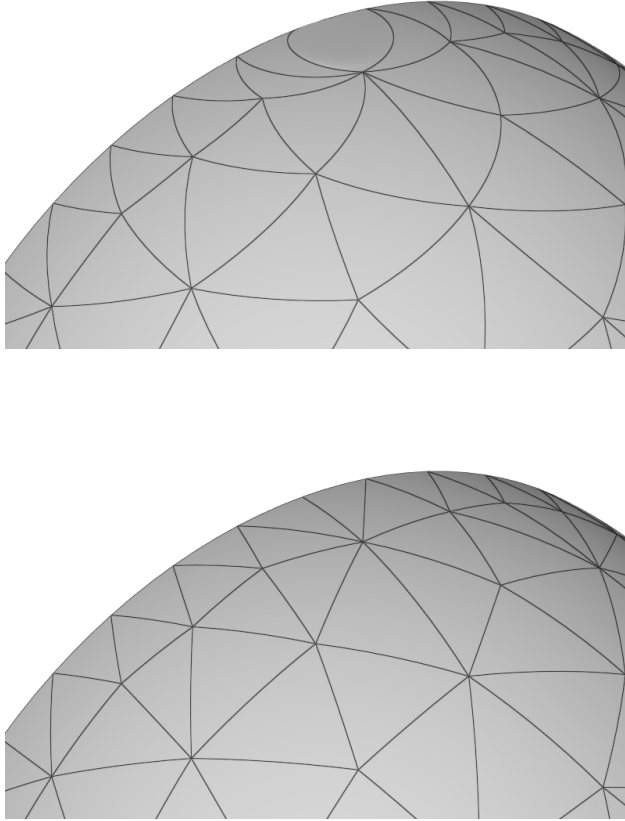
As for the face node, the case  $i = j = k = 1$ ,

$$\begin{aligned} 6P_{111} &= 27P_{111}^\ell - P_{300} - 3P_{210} - 3P_{120} - P_{030} \\ &\quad - 3P_{021} - 3P_{012} - P_{003} - 3P_{102} - 3P_{201}, \end{aligned}$$

which we compute using the previous.

Let us now compare the two approaches to constructing the Lagrange nodes: the direct evaluation and the projection. Fig. 5 illustrates the typical case of a half sphere mapped to from a rectangle in parametric space. It has two poles where quad edges have degenerated into a single point. This leads to points close to each other on the surface to have very distant parametric coordinates. In particular, edges of triangles close to these poles can be seen to be very curved (top figure) when the Lagrange control nodes were evaluated directly. Fortunately, this does not affect the Lagrange nodes which were projected from the straight positions as much. Indeed, this is less of a problem of ill-definition than one of strong variations: this is a region where points close in parametric space are sent to positions far from each other in physical space. This is not, however, truly a problem for projection, especially for edge nodes that are well in the interior of the face patch. However, in both cases but especially in the second, it is now possible to compute a surface normal and curvatures where the CAD previously did not allow.

There is room for improvement in the choice of the control points. For now, these are placed at the straight positions ( $P_{ijk}^\ell = \frac{i}{3}P_{300} + \frac{j}{3}P_{030} + \frac{k}{3}P_{003}$ ) in parametric space. It could be beneficial to optimize this initial placement with regards to geometric deviation,  $\mathcal{G}^1$  continuity or edge length in metric space with the  $P^3$  surface approximation metric in parametric space. Note that this  $P^3$  mesh is not guaranteed to be  $\mathcal{G}^1$  continuous contrarily to those constructed using the tangent plane method [22]. However, as the benchmarks in the following subsection show, this is of little consequence in practice.



**Figure 5:** Comparison of  $P^3$  meshes with Lagrange control nodes created using the direct evaluation approach (top) and the projection of the straight element approach (bottom) in the vicinity of the pole of a sphere.

### 2.3 Geometric primitives on the $P^3$ CAD surrogate

This mesh has been constructed with the objective of providing fast and robust projection on the surface as well as evaluation of derivatives. Indeed, when adapting the computational surface mesh, new vertices are created which must lie on the geometry. Likewise, surface optimization may call for normals or curvatures which are linked to, respectively, first and second derivatives of the surface parameterization. These two steps commonly fail on CAD systems. One common cause of failure for projection is the high tolerances present between patches: too close to patch intersections, the geometry is ill-defined. Another problem comes from the fact that it is carried out by gradient-based optimization — typically, Newton — in parameters space at the patch level. This means that, to project point  $P$  on the face with map-

ping  $\sigma$  over parametric domain  $D$ , it is the couple  $(u, v) = \min_D \{ \|\sigma - P\| \}$  that is sought and the (up to second) derivatives of  $\sigma$  are used. Therefore, projections may fail because of degeneracies of the mapping (such as  $\sigma$  mapping a boundary edge of  $D$  to a single point as for a cone) which lead to undefined behaviour of the derivatives.

Given a  $P^3$  element  $K$  and barycentric coordinates of some point on the triangle, computing derivatives is trivial due to the polynomial nature  $K$ . Projection becomes slightly more algorithmic, and proceeds in two steps. The lower level step consists in computing, for a given triangle  $K$ , the barycentric coordinates of the projected point onto  $K$ . The higher level strategy uses this step to move around elements according to the sign of these barycentric coordinates. Finding the barycentric coordinates of a point on a  $P^3$  element is relatively costly, since it requires several steps of a gradient-based optimization algorithm.  $P^1$  triangles, on the contrary, give exact barycentric coordinates in a single step using ratios of triangle areas. Denoting by  $bary_{deg}(P, K)$  a function returning the barycentric coordinates of point  $P$  in triangle  $K$  seen as a degree  $deg$  triangle, the projection algorithm can be summarized as follows:

Input: point  $P$ , initial guess  $K$

While not found:

For  $deg = (1, 3)$ :

While not found:

$\xi \leftarrow bary_{deg}(P, K)$

If  $\xi_1 > 0, \xi_2 > 0, \xi_3 > 0$ :

break

Else:

$K \leftarrow i$ -th neighbour of  $K$  s.t.  $\xi_i < 0$

The first iteration of the outer loop closes in on the correct element, using only cheaper projections on linear elements, so that very few (often one) steps are left to identify the correct  $P^3$  triangle in the second iteration. Some details were omitted for simplicity, such as usual search logic (marking elements so as to avoid repetition, stacking or sorting candidates when two barycentric coordinates are negative) and the fact that a guess for the normal is supplied to avoid finding a local minimum on the wrong side of a thinly folded surface and to distinguish the cases where the point lies outside of an open surface.

### 3. NUMERICAL EXAMPLES

#### 3.1 Benchmarks

We now turn to simple benchmarks of the  $P^3$  projection operator and of the  $P^3$  mesh construction. The geometry used in this section is the High-Lift Common Research wing Model (HL-CRM) with flaps used in the 4-th AIAA CFD High Lift Prediction Workshop. Figure 6 illustrates this geometry. The model contains 262 CAD faces and 700 CAD edges. This geometry is relatively complex with a number of degenerate patches: any triangular patches were originally mapped from a rectangular domain (green, orange domains in bottom figure).

##### 3.1.1 Projection speed and accuracy

For the first test, we loop over elements and generate  $N$  random points on each  $P^3$  triangle. We then call the projection algorithm with no knowledge of the solution and compare its return value to the expected point. Euclidean norm of the difference is used to compute  $\ell^2$  and  $\ell^\infty$  norms of the projection error. This is compared to a similar test on the CAD system, wherein points are generated on the geometry and then projected using a reasonable first guess (closest point in mesh). CAD projections are carried out using EGADSLite [23] built-in tools. Two meshes of 48 and 92 thousand triangles obtained by tessellating with surface error metrics for a geometric tolerance of 0.1 and 0.05 respectively are used. The only influence mesh coarseness should have on the results is in making the  $P^3$  triangles slightly flatter. Indeed, we are not measuring deviation from the  $P^3$  mesh to the geometry but rather the ability of the projection algorithm to recover a point that is exactly on the surface. Table 1 offers a summary of the results on these two meshes created from the same high-lift geometry. Both for the CAD and  $P^3$  tests and on either mesh,  $1M$  test points were generated to project. Projection errors are normalized for wing length. The first thing that stands out is that the  $P^3$  projection is in the order of 3 times faster than the projection on the CAD using EGADSLite. Projection quality is better using the CAD on average, with  $\ell^2$  errors for the  $P^3$  projection in the order of  $1e-8$  and in the order of  $1e-11$  for the CAD projection. EGADSLite CAD projections therefore tend to yield results in the order of 3 orders of magnitude times better than the  $P^3$  projection in its current state. However, looking at  $\ell^\infty$  errors, it appears that CAD projection is capable of more catastrophic failures, with a highest error of roughly 10% of the model's size on at least one point. This is not an isolated result, as similar behaviour has been observed on other cases. We believe the high average quality of CAD projections is due to the fact that NURBS are

mostly regular, except on patch boundaries when singularities exist. A number of these singularities exist on this model, such as on the bounding box (a half sphere with two degenerate edges at the poles) and on some triangular patches seen Fig. 6 with one degenerate edge. The  $P^3$  mesh, on the other hand, is regular and unbothered by these singularities, except potentially on construction. This explains that  $\ell^\infty$  errors on  $P^3$  projection remain more controlled. As for its worse  $\ell^2$  accuracy, the algorithm implemented to carry out  $P^3$  projection is a very rudimentary Newton descent with a fixed iteration count, no line search nor preconditioning. Using a more sophisticated algorithm would yield better results, as the optimization problem at hand is relatively simple.

# Triangles	48k	92k
$P^3$ err. $\ell^2$	$4.2e-8$	$2.1e-8$
$P^3$ err. $\ell^\infty$	$1.15e-2$	$8.3e-3$
$P^3$ CPU	1.15M p/sec	1.08M p/sec
CAD err. $\ell^2$	$6.7e-11$	$4.1e-11$
CAD err. $\ell^\infty$	$1.1e-1$	$1.1e-1$
CAD CPU	366k p/sec	390k p/sec

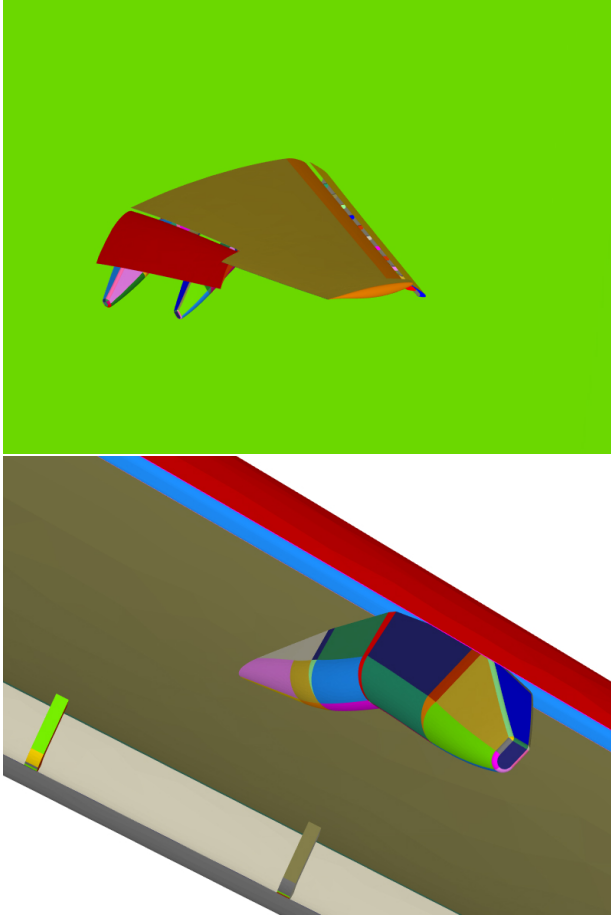
**Table 1:** Results for the first benchmark: projection of random points using the CAD system (EGADSLite) and the  $P^3$  CAD surrogate. CPU times given in projections per second (p/sec).  $10^6$  points were generated in both cases.

##### 3.1.2 Surface approximation error

Now that the  $P^3$  projection operator has been verified, we can move onto the second test involving geometric approximation. Here, we seek to evaluate the gap between the actual surface (CAD model) and the surface meshes ( $P^1$  and  $P^3$ ). There are two approaches to this. The first involves generating points on the  $P^3$  and  $P^1$  geometries and then projecting them onto the CAD model. The second does the opposite: points are generated on the CAD model and then projected onto the  $P^1$  and  $P^3$  surfaces. We chose the second approach because it is stabler, in that the CAD model is only used for evaluation and the mesh projection has been validated by that point.

To follow this approach, we proceed on a per-face basis and require a tessellation of the trimmed parametric domain of each CAD face. This is accomplished easily at this stage. Indeed, the surface mesher keeps records of which CAD faces and edges created which mesh vertices and at which parameter values. In other words, for any vertex  $P_i$  of the surface mesh, it is trivial to retrieve the  $\sigma$  and  $\xi_i$  for which  $\sigma(\xi_i) = P_i$ . Now, given the set of triangles and vertices ( $P_i$ ) that discretize a given CAD face, the desired tessellation of the trimmed parameters domain is given by the mesh with the same connectivity and vertices  $\xi_i$ . We denote





**Figure 6:** The high-lift wing model with flaps.

$(K_i)_{1 \leq i \leq N}$  the  $N$  triangles of this tessellation and  $\Pi^k$  the projection operator on the degree  $k$  mesh. We seek to evaluate the quantities

$$E_p^k = \|\sigma - \Pi^k \sigma\|_{L^p(\cup_{i=1}^N K_i)} \quad (1)$$

for  $p$  either 2 or  $\infty$ . In the case of  $p = 2$ ,

$$\begin{aligned} E_2^k &= \sum_{i=1}^N \int_{K_i} \|\sigma(\xi) - \Pi^k(\sigma(\xi))\| d\xi \\ &= \sum_{i=1}^N |J_{K_i}| \int_{\hat{K}} \|\sigma(F_K(\hat{\xi})) - \Pi^k(\sigma(F_K(\hat{\xi})))\| d\hat{\xi} \end{aligned} \quad (2)$$

where  $F_K : \hat{K} \mapsto K$  is the reference-to-physical mapping of the triangle  $K$  and  $J_K$  the determinant of its Jacobian. Note that, since we are dealing with elements defined in parametric space, this quantity is well-defined since elements are in  $\mathbb{R}^2$ . Furthermore, even in the case of a  $P^3$  triangle, the Jacobian is constant because we have constructed the  $P^3$  triangles to

be straight in parametric space (their mapping degenerates into the linear one) and it is only in physical space that they are curved. These individual integrals are then evaluated by simple uniform quadrature. The reason for this choice is that we seek to evaluate this integral with a great degree of precision and such a scheme is very simple to converge, though at a slower pace. 45 quadrature points were used. As for the  $L^\infty$  error, we simply take the maximum absolute value at the quadrature points, needing no further computations. Greater accuracy on the  $L^\infty$  error is another advantage of a quadrature scheme using more points. We also define the errors on the normal directions

$$\partial E_p^k = \|\mathbf{n}_\sigma(u, v) - \mathbf{n}^k(\Pi^k \sigma)\|_{2, L^p(\cup_{i=1}^N K_i)} \quad (3)$$

where  $\mathbf{n}_\sigma(u, v) = (\partial_u \sigma \wedge \partial_v \sigma) / \|\partial_u \sigma \wedge \partial_v \sigma\|$  is the normalized normal vector computed using the CAD patch and  $\mathbf{n}^k(X)$  is the normalized normal at point  $X$  on the  $P^k$  mesh. Furthermore, errors are the normalized  $\tilde{E}_p^k = E_p^k C^k$  where  $C^k$  is the mesh complexity for a given degree given by

$$\begin{aligned} C^1 &= 3N_T + 3\rho N_P^1 \\ C^3 &= 10N_T + 3\rho(N_T + N_P^1 + 2N_E) \end{aligned} \quad (4)$$

where  $N_T$  denotes the number of triangles in the mesh,  $N_P^1$  the number of vertices in the  $P^1$  mesh,  $N_E$  the number of edges.  $\rho$  is the real to integer storage cost ratio, with  $\rho = 2$  in our case given that 32 bit integers and 64 bit reals (double precision) were used. The same goes for the errors on normals  $\partial \tilde{E}_p^k$ . Finally, we normalize by the total area of the surface, estimated from the  $P^1$  mesh.

Results are summarized in Tab. 2. Complexities for the  $P^1$  and  $P^3$  meshes show that, in both cases, the  $P^3$  mesh was upwards of 6 times heavier in memory than the  $P^1$  mesh for the same number of triangles. However, looking at the ratios of errors, the  $P^1$  mesh is clearly much worse than the  $P^3$  mesh for the same number of degrees of freedom. On the coarser mesh, the  $P^1$  mesh was 72 times worse in  $L^2$  norm and 7.5 times worse in  $L^\infty$  norm. These figures jump to a 1900, resp. 83, times worse  $P^1$  mesh at the same number of degrees of freedom with the finer mesh. This alone heavily favours the  $P^3$  CAD surrogate for geometric projection. But looking at the errors on normals, it comes with no surprise that the  $P^3$  mesh is thousands of times more accurate than the  $P^1$  mesh at a given number of degrees of freedom. Even more so on the coarser mesh, where the piece-wise constant normals given by the  $P^1$  mesh are no match for the smooth normals of the  $P^3$  mesh (factor  $\times 20000$  more accurate on average and in the worst case).

$N_T/C^1/C^3$	48k / 290k / 1.8M	92k / 550k / 3.4M
$\tilde{E}_2^1$	5.7e2	41
$\tilde{E}_\infty^1$	5.7e-1	4.5e-2
$\tilde{E}_2^3$	7.9	2.2e-2
$\tilde{E}_\infty^3$	7.7e-2	5.4e-4
$\tilde{E}_2^1/\tilde{E}_2^3$	72	1.9e3
$\tilde{E}_\infty^1/\tilde{E}_\infty^3$	7.5	83
$\partial\tilde{E}_2^1$	9.0	2.2
$\partial\tilde{E}_\infty^1$	5.9e-3	3.1e-3
$\partial\tilde{E}_2^3$	4.7e-4	4.5e-4
$\partial\tilde{E}_\infty^3$	3.7e-7	3.2e-7
$\partial\tilde{E}_2^1/\partial\tilde{E}_2^3$	1.9e4	4.9e3
$\partial\tilde{E}_\infty^1/\partial\tilde{E}_\infty^3$	1.6e4	9.8e3

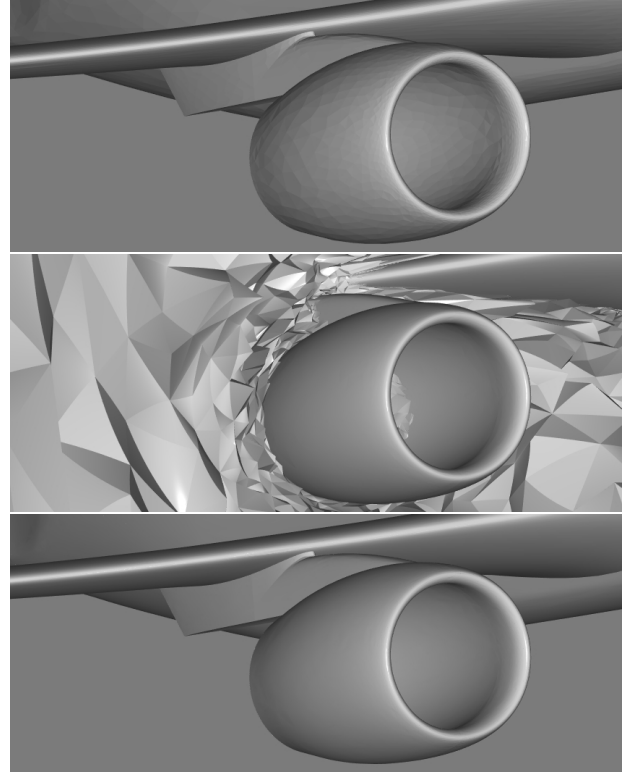
**Table 2:** Normalized errors for the  $P^1$  and  $P^3$  meshes. Errors are given for two coarseness levels: 48000 and 92000 triangles.

### 3.1.3 $P^2$ volume meshes

Finally, we present a minor (with regards to adaptation as a whole) use of these  $P^3$  CAD surrogate meshes, namely as a geometric support to curve  $P^2$  meshes. Acknowledgement of the importance of curved volume meshes dates back to the 70s with the proof that optimal convergence of high-order methods is only possible with a curved boundary in the case of elliptic problems [24, 25] and later for hyperbolic problems, where physical features are lost on  $P^1$  boundaries [26]. Unlike CAD surrogate meshes, these curved surface meshes are constrained by the validity of inciding tetrahedra. Indeed, they are comprised of triangles that are faces of volume elements whose Jacobian determinants must remain positive. A volume mesh curving technique based on minimizing edge lengths in the metric field [27] could be extended to surface meshes by parameterizing Lagrange node position of  $P^2$  edges as barycentric coordinates on  $P^3$  CAD surrogate mesh elements. This takes the constraint that the Lagrange node must lie on the surface into account at a lower level than by letting it be any point in space that is later projected on the surface. Not only does this reduce the dimension of the edge length minimization problem from 3 to 2 variables, it also makes for more accurate derivatives of the cost function. This would be much more complicated to do on the CAD, where the parameterizations are more sophisticated, not to mention the problems already cited before.

Figure 7 illustrates results on the 3rd High-Lift CRM. This is a whole-body model with fewer features. The  $P^1$  mesh was obtained as part of the high-lift drag prediction workshop. It is the result of adaptation using AMG/fefflo.a [28] for mesh modifications and the solver Wolf [29]. The  $P^3$  surface mesh was then cre-

ated and used to project Lagrange nodes of the  $P^2$  surface mesh.  $P^2$  volume edges were then curved using the metric field by minimizing edge lengths. Without too much surprise, the  $P^2$  surface mesh inherits the good properties of the  $P^3$  CAD surrogate.



**Figure 7:** High-Lift CRM of the 3rd AIAA CFD High Lift Prediction Workshop meshes of degrees 1, 2 and 3 from top to bottom. The  $P^1$  mesh (top) is the result of adaptation. The  $P^3$  mesh (bottom) was elevated from it using the method presented here. The  $P^2$  mesh (middle) was elevated from the  $P^1$  mesh using projection on the  $P^3$  mesh rather than the CAD. Volume elements (visible in the cut plane) have positive Jacobian determinants despite the clearly curved surface.

## 3.2 Adaptation convergence

In the previous section, we have shown that the  $P^3$  CAD surrogate mesh is accurate as desired and that the geometric primitives are fast and robust. We now present a more pragmatic test, based on exhibiting convergence of the remeshing algorithm with an analytical boundary-layer metric while using the  $P^3$  CAD surrogate for point projections. This metric is of the form:

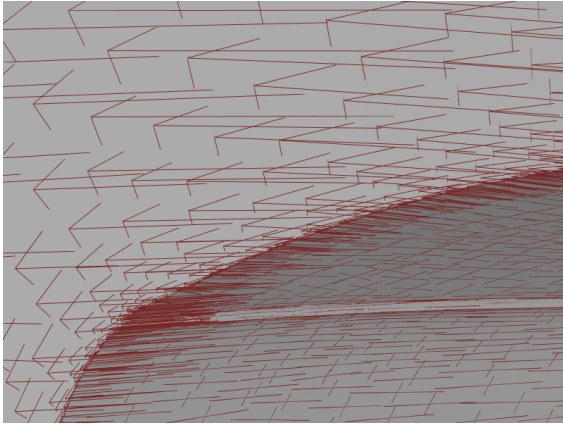
$$\mathcal{M}(P) = \mathcal{R}\Lambda\mathcal{R}^T$$

with

$$\Lambda = \text{diag}(h_M^{-2}, h_M^{-2}, \min(h_m^{-2}, \max(\exp(1/\|P - \Pi P\|), h_M^{-2})))$$

$$\text{and } \mathcal{R} = (\tau_1(\Pi P) \quad \tau_2(\Pi P) \quad \mathbf{n}(\Pi P))$$

where  $\Pi P$  denotes the projection of  $P$  onto the surface,  $\tau_i$  are unit tangent vectors and  $\mathbf{n}$  a unit normal at the surface. Finally,  $h_m$  and  $h_M$  are, respectively, the minimum and maximum metric sizes. This metric field is anisotropic with the smallest size along the normal direction to the surface. This size converges to the minimum admissible size as one goes closer to the surface, leading to a mesh with that prescribed size extruding from the surface. This metric field is illustrated in Fig. 8.



**Figure 8:** Analytical metric field used. Metrics are represented by their eigenvectors pondered by the square root of the inverse of their eigenvalues (characteristic sizes).

This is a challenging case in that very small edges are liable to appear in the vicinity of patch junctions, where the geometry is ill-defined by the CAD system. The converged adapted meshes must display the proper anisotropy even close to the surface. Furthermore, geometric approximation error must not increase during adaptation. Otherwise, this would mean that the surface mesh is converging to the wrong geometry. We will compare two adaptation runs: the first uses the  $P^3$  CAD surrogate, the second the CAD model. Adaptation is carried out by the automatic metric-based remesher AMG/fefflo.a [28]. The overall adaptation proceeds as follows:

While target mesh complexity not reached

    Compute metric field with increased complexity target

    Adapt mesh to metric field

This is necessary because, despite the fact that the chosen metric is analytical, the metric field is only

	$P^3$ CAD surrogate	CAD model
CPU time	9m18s	9m16s
Unit Edge %	91.4	91.5
$Q_{max}$	74	97
Final Geo. err.	3.9e-6	5.5e-6

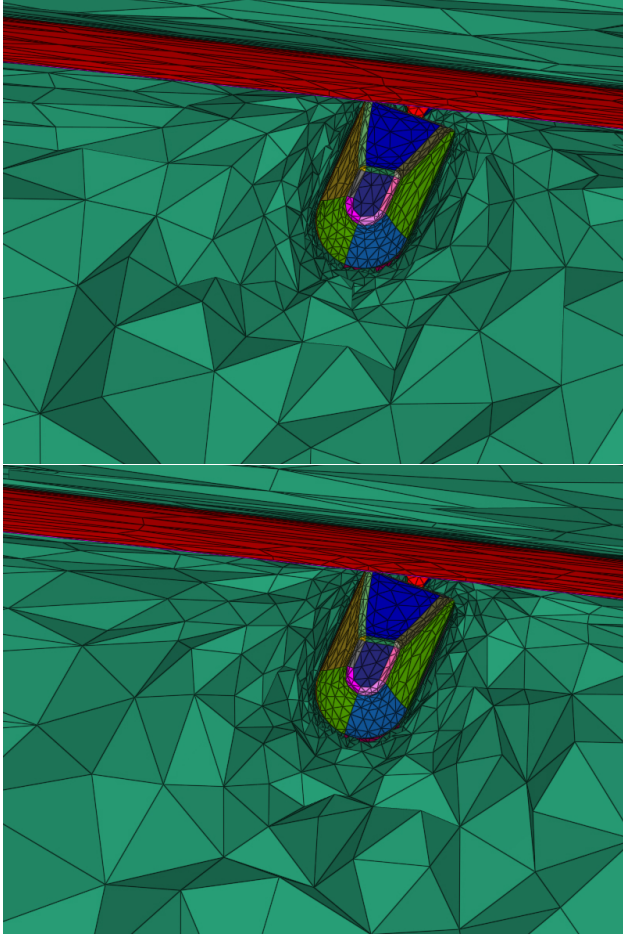
**Table 3:** Adaptation metrics: CPU time for the entire process, overall mesh quality (unit edges), maximum surface quality, final geometric approximation error of the adapted mesh.

known at the vertices of the input mesh and is therefore an approximation of the desired metric field. As such, even starting at the correct complexity, several iterations would be necessary. Starting from a lower complexity target affords faster convergence since individual executions of the remeshing algorithm are faster to execute. In our case, 13 iterations were carried out.

Fig. 9 illustrates the adapted results in both cases. CPU times and mesh quality metrics are given in Tab. 3. Unit edges are those for which their length in the metric field lies in  $[1/\sqrt{2}, \sqrt{2}]$ . The quality  $Q$  of a triangle  $K$  is

$$Q(K) = \frac{1}{2\sqrt{3}} \frac{\sum l_i^2}{\mathcal{A}(K)}$$

where the  $l_i$  are the lengths of the edges,  $\mathcal{A}(K)$  is the area of  $K$ , and both quantities are computed in the metric field. The scaling factor is such that the minimum of  $Q$  over all possible triangles is 1. This corresponds to unit triangles in the metric, whereas as  $Q$  goes to infinity, triangles are flatter and less unit in the metric. We then define  $Q_{max}$  as the maximum quality over all triangles in the mesh. CPU times are extremely close. This is simply a consequence of the fact that surface projection is not very significant in overall adaptation CPU time. Indeed, the majority of mesh vertices lie in the volume and even for surface points the projection times represent at most one tenth of insertion time. As an order of comparison, our algorithm inserts between 25 and 40 thousand points per second depending on compilation flags, whereas point projection proceeded at a rate of 300 thousand per second even for the slowest case, direct CAD projection. Proportion of unit edges in the mesh and maximum surface element quality are within reasonable values. This in itself is not enough to conclude that  $P^3$  projection is sufficient. Indeed, we must now compare geometric approximation errors and ensure that the adapted surface mesh is not degraded with regards to the initial mesh. Geometric error is reported in  $L^\infty$  norm as computed in (2). In both cases, the final mesh exhibits an error under the prescribed geometric error of  $2e-3$  by a large margin.



**Figure 9:** Detail of resulting adapted meshes using the  $P^3$  CAD surrogate (top) and the CAD model (bottom) for the analytical boundary layer metric.

## 4. CONCLUSION

In this paper, we have exhibited a very simple extension to a parametric meshing algorithm that produces polynomial  $P^3$  meshes from CAD objects, *i.e.* continuous representations riddled with singularities, degenerating features and large tolerances at intersections. These meshes are constructed in a preprocessing step and are then taken as input instead of the CAD file on each adaptation call. Tolerances at junctions between NURBS patches were managed by treating the problem on a discrete level. To do this, physical CAD edges are first discretized. This yields a set of vertices and edges of  $\mathbb{R}^3$  which are then projected onto the parametric patches of neighbouring NURBS faces. This set of edges is then used to trim the NURBS patch instead of the CAD supplied parametric trimming curve, eliminating mismatches between points and edges generated by each NURBS face.

Two approaches for creating  $P^3$  meshes from an initial  $P^1$  surface and a CAD model were described. The first only involved CAD evaluations, by setting the parametric coordinates of the new control points at the thirds of those of edge extremities (and triangle vertices, in the case of face control nodes). The second required CAD projections, since  $P^3$  triangles were first made straight in physical space and then projected onto parametric space. In the absence of further optimization, the second approach is more stable, as the first method easily produces bad high-order nodes close to CAD discontinuities such as the poles of a sphere. Resulting  $P^3$  meshes are lightweight and allow for fast projection on the surface. They improve greatly on geometric deviation over  $P^1$  meshes as well as derivative computations which can be in the order of  $10^4$  times more accurate for the same storage cost. Furthermore, derivatives are always defined on a  $P^3$  Bézier mesh, which they are not necessarily on trimmed NURBS. Point projections are in the order of 3 times faster on  $P^3$  meshes as well, by using the implicit  $P^1$  mesh as an accelerator. Finally, the fact that the CAD model is only used in the preprocessing step reduces the frequency at which CAD errors may occur in the adaptation process. One common occurrence is a shock that spans parts of the geometry where NURBS meet. At these junctions, the CAD object has large tolerances, essentially meaning these interfaces are fuzzy under some scale. This scale tends to be much larger than that of desired element size at shocks, limiting potential anisotropy and robustness of adaptation. This cannot possibly happen on the  $P^3$  mesh which is watertight by construction.

These virtues of the  $P^3$  geometric representation were put to the test through a full adaptation run on the High-Lift model. Comparing between adaptation using CAD projection and using  $P^3$  projection, we have shown that the  $P^3$  surface mesh is sufficiently accurate to carry out mesh adaptation (including surface adaptation) on complex geometries with a strongly anisotropic metric field. Another advantage of using  $P^3$  meshes for surface adaptation is that the CAD is not necessary. Although the techniques shown here involve a CAD,  $P^3$  meshes can be constructed from a straight mesh by estimating normals at the vertices and enforcing  $\mathcal{G}^1$  continuity [22].

The fidelity of the CAD surrogate meshes can be improved in at least two ways. The first is by optimizing the Lagrange node placement of  $P^3$  triangles so as to minimize surface approximation error. This has not been done at all, Lagrange nodes are simply placed on the straight element at the thirds and then projected onto the geometry. There is a direct link between Bézier nodes and tangent planes of  $P^3$  triangles (the tangent plane at  $P_{300}$  is the span of the vectors  $P_{210}$  and  $P_{120}$ , for instance) which could help devise

a procedure to fit the derivatives of the surface mapping up to the third order with little cost. An approach using the geometric approximation metric field to seek geodesics in parameter space to avoid the phenomenon illustrated in figure 5 is another possibility, albeit perhaps more costly. Since the  $P^3$  mesh creation step is accomplished once per full adaptation, we expect such optimizations would not prove too costly in the grand scheme of things, while possibly enabling coarser  $P^3$  meshes to approximate the surface under the tolerance. Finally, these meshes could be improved by increasing their degree. The main difficulty with higher order meshes would be in extending the aforementioned optimization strategies. The second way these CAD surrogate meshes could be improved involves replacing parts of the mesh with pieces of the CAD, such as rational Bézier triangles. These could be extracted from the CAD's NURBs patches in such a way that they would be exact representations of the original parameterization, while being an intermediary step towards a discrete mesh helping projection. This could provide a discrete-exact description of the geometry, with CAD faces being exactly represented in their interior, and approximated close to the trimming curve by a discrete  $P^3$  mesh. Likewise, it is possible that some singularities arising from degenerating CAD edges could be fixed this way.

## References

- [1] Loseille A., Alauzet F. “Continuous mesh framework part I: well-posed continuous interpolation error.” *SIAM Journal on Numerical Analysis*, vol. 49, no. 1, 38–60, 2011
- [2] Loseille A., Alauzet F. “Continuous mesh framework part II: validations and applications.” *SIAM Journal on Numerical Analysis*, vol. 49, no. 1, 61–86, 2011
- [3] Loseille A., Dervieux A., Alauzet F. “Fully anisotropic goal-oriented mesh adaptation for 3D steady Euler equations.” *Journal of Computational Physics*, vol. 229, no. 8, 2866 – 2897, 2010
- [4] Loseille A., Menier V. “Serial and parallel mesh modification through a unique cavity-based primitive.” *Proceedings of the 22nd International Meshing Roundtable*, pp. 541–558. Springer, 2014
- [5] Löhner R., Baum J.D. “Adaptive h-refinement on 3D unstructured grids for transient problems.” *International Journal for Numerical Methods in Fluids*, vol. 14, no. 12, 1407–1419, 1992
- [6] Babuska I., Szabo B.A., Katz I.N. “The p-version of the finite element method.” *SIAM Journal on Numerical Analysis*, vol. 18, no. 3, 515–545, 1981
- [7] Kompenhans M., Rubio G., Ferrer E., Valero E. “Comparisons of p-adaptation strategies based on truncation-and discretisation-errors for high order discontinuous Galerkin methods.” *Computers & Fluids*, vol. 139, 36–46, 2016
- [8] Ceze M., Fidkowski K.J. “Anisotropic hp-adaptation framework for functional prediction.” *AIAA Journal*, vol. 51, no. 2, 492–509, 2013
- [9] Dolejsi V., Ern A., Vohralík M. “hp-adaptation driven by polynomial-degree-robust a posteriori error estimates for elliptic problems.” *SIAM Journal on Scientific Computing*, vol. 38, no. 5, A3220–A3246, 2016
- [10] Huerta A., Angeloski A., Roca X., Peraire J. “Efficiency of high-order elements for continuous and discontinuous Galerkin methods.” *International Journal for Numerical Methods in Engineering*, vol. 96, no. 9, 529–560, 2013
- [11] Vanharen J. *High-order numerical methods for unsteady flows around complex geometries*. Ph.D. Thesis, Université de Toulouse, 2017
- [12] Farin G.E., Farin G. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, 2002
- [13] Piegl L., Tiller W. *The NURBS book*. Springer Science & Business Media, 1996
- [14] Marussig B., Hughes T.J. “A review of trimming in isogeometric analysis: Challenges, data exchange and simulation aspects.” *Archives of Computational Methods in Engineering*, vol. 25, no. 4, 1059–1127, 2018
- [15] Beall M.W., Walsh J., Shephard M.S. “Accessing CAD Geometry for Mesh Generation.” *Imr*, pp. 33–42. 2003
- [16] Park M.A., Haimes R., Wyman N.J., Baker P.A., Loseille A. “Boundary Representation Tolerance Impacts on Mesh Generation and Adaptation.” *AIAA AVIATION 2021 FORUM*, p. 2992. 2021
- [17] Sheffer A., Bercovier M., BLACKER T., Clements J. “Virtual topology operators for meshing.” *International Journal of Computational Geometry & Applications*, vol. 10, no. 03, 309–331, 2000
- [18] Frey P. “About Surface Remeshing.” *9th International Meshing Roundtable*. Sandia National Laboratories, 2000
- [19] Feuillet R., Coulaud O., Loseille A. “Anisotropic error estimate for high-order parametric surface mesh generation.”

- [20] Loseille A., Alauzet F. “Optimal 3D highly anisotropic mesh adaptation based on the continuous mesh framework.” *Proceedings of the 18th International Meshing Roundtable*, pp. 575–594. Springer, 2009
- [21] Frey P.J., Alauzet F. “Anisotropic mesh adaptation for CFD computations.” *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 48-49, 5068–5082, 2005
- [22] Vlachos A., Peters J., Boyd C., Mitchell J.L. “Curved PN triangles.” *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pp. 159–166. 2001
- [23] Haimes R., Dannenhoffer J. “EGADSLite: A Lightweight Geometry Kernel for HPC.” *2018 AIAA Aerospace Sciences Meeting*, p. 1401. 2018
- [24] Ciarlet P.G., Raviart P.A. “The combined effect of curved boundaries and numerical integration in isoparametric finite element methods.” *The mathematical foundations of the finite element method with applications to partial differential equations*, pp. 409–474. Elsevier, 1972
- [25] Lenoir M. “Optimal isoparametric finite elements and error estimates for domains involving curved boundaries.” *SIAM Journal on Numerical Analysis*, vol. 23, no. 3, 562–580, 1986
- [26] Bassi F., Rebay S. “High-order accurate discontinuous finite element solution of the 2D Euler equations.” *Journal of Computational Physics*, vol. 138, no. 2, 251–285, 1997
- [27] Loseille A., Rochery L. “Developments on the  $P^2$  cavity operator and Bézier Jacobian correction using the simplex algorithm.” *AIAA SCITECH 2022 Forum*, p. 0389. 2022
- [28] Loseille A. “Chapter 10 - Unstructured Mesh Generation and Adaptation.” *Handbook of Numerical Methods for Hyperbolic Problems*, vol. 18 of *Handbook of Numerical Analysis*, pp. 263 – 302. Elsevier, 2017
- [29] Alauzet F., Frazza L. “3D RANS anisotropic mesh adaptation on the high-lift version of NASA’s Common Research Model (HL-CRM).” *AIAA Aviation 2019 Forum*, p. 2947. 2019



# BISECTING WITH OPTIMAL SIMILARITY BOUND ON 3D UNSTRUCTURED CONFORMAL MESHES

Guillem Belda-Ferrín<sup>1</sup>

Eloi Ruiz-Gironés<sup>1</sup>

Xevi Roca<sup>1,2</sup>

<sup>1</sup>*Computer Applications in Science and Engineering,  
Barcelona Supercomputing Center - BSC, 08034 Barcelona, Spain*

<sup>2</sup>*Corresponding author: xevi.roca@bsc.es*

## ABSTRACT

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.

**Keywords:** bisection, marked bisection, newest vertex bisection

## 1. INTRODUCTION

In adaptive finite element analysis, unstructured tetrahedral meshes have to be locally adapted. To this end, one needs to perform local mesh modifications. One successful modification is to bisect the required tetrahedra. This bisection operation splits a tetrahedron 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 tetrahedra. The quality of these tetrahedra depends on the criterion to select refinement edges. This edge selection is commonly based on choosing either the longest edge [1, 2, 3, 4] or the newest vertex [5, 6].

The self-similarity of the newest vertex bisection [5, 6] has been favored over other bisection-based refinements in many three-dimensional applications. Specifically, in adaptive applications [7, 8, 9, 10] where it is possible to start with a three-dimensional reflected mesh [9, 10, 11, 12]. This preference is so since on re-

flected meshes local refinement with newest vertex bisection has the minimum mesh quality bounded. This bound is a consequence of the bound in the number of similarity classes of mesh tetrahedra. That is, for each initial tetrahedron, successive newest vertex bisection does not generate more than 36 different similarity classes [13]. This number of classes is the smallest bound known for a three-dimensional bisection method.

Unfortunately, this optimal bound has not been met in adaptive applications requiring complex three-dimensional geometries. This lack is so since practical methods to extract a reflection structure are limited to specific meshes. For instance, meshes generated using the Coxeter-Freudenthal-Kuhn algorithm [14, 15, 16] or meshes where all the edges have an even number of incident tetrahedra [9, 10]. Currently, there is no method known to extract a strong reflection structure from an arbitrary three-dimensional unstructured conformal mesh [8, 10, 13, 11, 12].

For three-dimensional unstructured conformal meshes,

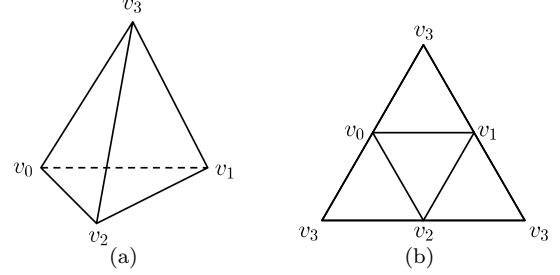
there are alternative bisection methods with sub-optimal similarity bound [17, 7, 18, 19, 13]. All these methods lead to analogous locally refined conformal meshes. Nevertheless, Arnold *et al.* [13] establish a key connection between Maubach’s newest vertex bisection [8] and marked bisection [13]. Marked 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 [13] 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’s newest vertex bisection [8] under specific conditions [13].

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, for some conformingly marked meshes, marked bisection behaves as the newest vertex bisection [13]. 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 [13]. 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 work 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



**Figure 1:** Representations of a tetrahedron composed of the vertices  $v_1$ ,  $v_2$ ,  $v_3$ , and  $v_4$ : (a) volumetric; and (b) planar.

guarantee conformingly marked meshes by checking that we fulfill the sufficient conditions for tetrahedral meshes stated in [13]. To illustrate the application, we implement the refine to conformity marked bisection [13] 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.

The rest of the paper is structured as follows. In Section 2, we state the problem. In Section 3, we propose a marking process and show that it generates conformingly marked meshes. Next, in Section 4, we detail the adaptation of Arnold’s bisection algorithm to our proposed marking process. In Section 5, we present several examples to show the features of the proposed method. Finally, in Section 6, we detail the conclusions and the future work.

## 2. 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.

### 2.1 Preliminaries: definitions

A *simplex* is the convex hull of  $n+1$  points  $p_0, \dots, p_n \in \mathbb{R}^n$  that do not lie in the same hyper-plane. We denote it as  $\sigma = \text{conv}(p_0, p_1, \dots, p_n)$ . 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  $\sigma = (v_0, v_1, \dots, v_n)$ . We have an application  $\Pi$  that maps each identifier  $v_i$  to the corresponding point such that  $\Pi(v_i) = p_i$ . In our application, we are interested in tetrahedra, three-dimensional simplices. Herein, as in [13], we represent volumetric tetrahedra composed of the vertices  $v_1$ ,  $v_2$ ,  $v_3$ , and  $v_4$ , see Figure 1(a), in the plane by cutting

and unfolding the corresponding triangular faces, see Figure 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  $\tau$ , respectively. We denote the faces, edges and vertices with the letters  $\kappa$ ,  $e$  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  $\tau$  with the opposite face to a vertex of  $\tau$ . 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  $\kappa_0 = (v_1, v_2, v_3)$ .

A *mesh*,  $\mathcal{T}$ , associated to an open set  $\Omega \in \mathbb{R}^n$  is a finite collection of mutually disjoint tetrahedra such that

$$\bar{\Omega} = \bigcup_{\tau \in \mathcal{T}} \tau.$$

A tetrahedral mesh is *conformal* if, for any  $\tau_1, \tau_2 \in \mathcal{T}$ ,  $\tau_1 \cap \tau_2$  is either empty, or a common edge, or a common triangle. We say that two tetrahedra  $\tau_1$  and  $\tau_2$  are *neighbors* if they share a common triangular face.

## 2.2 Preliminaries: marked bisection

Arnold *et al.* [13] 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*  $e_\tau$  is the edge of  $\tau$  to be bisected. Since an edge is shared by two triangular faces of the tetrahedron, the triangular faces that contain  $e_\tau$  are the *refinement faces* of  $\tau$ . 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  $\kappa_i$  has a refinement edge  $e_{\kappa_i}$ . Particularly, the refinement edge  $e_\tau$  is the same as the  $e_{\kappa_i}$  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 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 2(a), and 2(b), respectively.

---

### Algorithm 1 Refining a subset of a mesh.

---

**input:** Mesh  $\mathcal{T}$ , set of element  $\mathcal{S} \subset \mathcal{T}$  to refine

**output:** ConformalMarkedMesh  $\mathcal{T}_2$

```

1: function refineMesh( $\mathcal{T}, \mathcal{S}$ )
2:    $\mathcal{T}_1 = \text{markMesh}(\mathcal{T})$ 
3:    $\mathcal{T}_2 = \text{localRefine}(\mathcal{T}_1, \mathcal{S})$ 
4:   return  $\mathcal{T}_2$ 
5: end function
```

---

- Adjacent,  $A$ : the marked edges are adjacent to the refinement edge but are not coplanar, see Figures 2(c).
- Mixed,  $M$ : one marked edge is adjacent to the refinement edge, and the other is opposite, see Figures 2(d).
- Opposite,  $O$ : both marked edges are opposite to the refinement edge, see Figures 2(e).

These tetrahedron types are the nodes of the directed graph determining the marked bisection sequence, see Figure 3.

Now, we can introduce the definition of *marked tetrahedron*, which is a modification of the one detailed in [13]. Herein, a marked tetrahedron is the 5-tuple

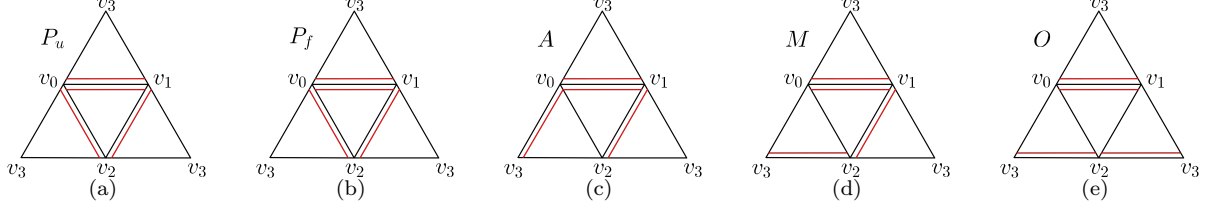
$$\rho = (\tau, e_\tau, e_{\kappa_1}, e_{\kappa_2}, t),$$

where  $\tau$  is a tetrahedron,  $e_\tau$  is the refinement edge,  $e_{\kappa_1}$  and  $e_{\kappa_2}$  are the marked edges of the non-refinement faces, and  $t$  is the tetrahedron type.

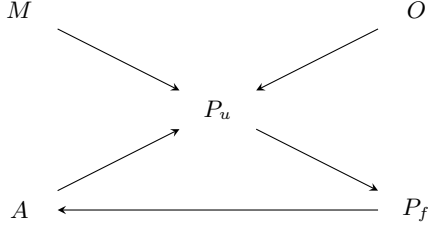
A mesh is *marked* if all its tetrahedra are marked. A marked conformal mesh is *conformingly marked* if each triangular face has a unique marked edge. That is, a triangular face shared by two tetrahedra has the same marked edge from both sides. Accordingly, shared triangular faces are bisected in the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

*Remark 2.1* (Conditions to conformingly mark). To guarantee that a conformal mesh is conformingly marked, Arnold *et al.* [13] 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 1, 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, first refines a set of tetrahedra and then calls a recursive refine-to-conformity strategy. This strategy, Algorithm 3, terminates when



**Figure 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.



**Figure 3:** Directed graph of tetrahedron types for marked bisection.

---

**Algorithm 2** Local refinement of a marked mesh.

---

**input:** ConformalMarkedMesh  $\mathcal{T}$  and  $\mathcal{S} \subset \mathcal{T}$   
**output:** ConformalMarkedMesh  $\mathcal{T}_0$

- 1: **function** localRefine( $\mathcal{T}, \mathcal{S}$ )
- 2:    $\bar{\mathcal{T}} = \text{bisectTetrahedra}(\mathcal{T}, \mathcal{S})$
- 3:    $\mathcal{T}_0 = \text{refineToConformity}(\bar{\mathcal{T}})$
- 4:   **return**  $\mathcal{T}_0$
- 5: **end function**

---



---

**Algorithm 3** Refine-to-conformity a marked mesh.

---

**input:** MarkedMesh  $\mathcal{T}$   
**output:** MarkedMesh  $\mathcal{T}'$  without hanging nodes

- 1: **function** refineToConformity( $\mathcal{T}$ )
- 2:    $\mathcal{S} = \text{getHangingNodes}(\mathcal{T})$
- 3:   **if**  $\mathcal{S} \neq \emptyset$  **then**
- 4:      $\bar{\mathcal{T}} = \text{bisectTetrahedra}(\mathcal{T}, \mathcal{S})$
- 5:      $\mathcal{T}' = \text{refineToConformity}(\bar{\mathcal{T}})$
- 6:   **else**
- 7:      $\mathcal{T}' = \mathcal{T}$
- 8:   **end if**
- 9:   **return**  $\mathcal{T}'$
- 10: **end function**

---

successive bisection leads to a conformal mesh. Both algorithms use marked bisection to refine a set of elements, see Algorithm 4.

*Remark 2.2* (Optimal similarity bound). If the conformingly marked mesh is composed only of unflagged

---

**Algorithm 4** Bisect a set of tetrahedra.

---

**input:** MarkedMesh  $\mathcal{T}$ , SetSimplices  $\mathcal{S}$   
**output:** MarkedMesh  $\mathcal{T}_1$

- 1: **function** bisectTetrahedra( $\mathcal{T}, \mathcal{S}$ )
- 2:    $\mathcal{T}_1 = \emptyset$
- 3:   **for**  $\rho \in \mathcal{T}$  **do**
- 4:     **if**  $\rho \in \mathcal{S}$  **then**
- 5:        $\rho_1, \rho_2 = \text{bisectTet}(\rho)$
- 6:        $\mathcal{T}_1 = \mathcal{T}_1 \cup \rho_1$
- 7:        $\mathcal{T}_1 = \mathcal{T}_1 \cup \rho_2$
- 8:     **else**
- 9:        $\mathcal{T}_1 = \mathcal{T}_1 \cup \rho$
- 10:    **end if**
- 11:   **end for**
- 12:   **return**  $\mathcal{T}_1$
- 13: **end function**

---

planar or adjacent tetrahedra, marked bisection does not generate more than 36 similarity classes [13].

## 2.3 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 2.2. Specifically, starting on the unflagged planar mesh  $\mathcal{T}_1$ , if we locally refine a set of elements  $\mathcal{S}$ , 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.

## 3. SOLUTION: CONFORMINGLY MARKING AS UNFLAGGED 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

---

**Algorithm 5** Marking as unflagged planar.

---

**input:** Tetrahedron  $\tau$ 
**output:** MarkedTetrahedron  $\rho$ 

```

1: function markTetrahedron( $\tau$ )
2:    $e_\tau = \text{consistentBisectionEdge}(\tau)$ 
3:    $(v_0, v_1) = e_\tau$ 
4:    $\kappa_1 = \text{oppositeFace}(\tau, v_0)$ 
5:    $\kappa_2 = \text{oppositeFace}(\tau, v_1)$ 
6:    $e_{\kappa_1} = \text{consistentBisectionEdge}(\kappa_1)$ 
7:    $e_{\kappa_2} = \text{consistentBisectionEdge}(\kappa_2)$ 
8:    $t = P_u$   $\triangleright$  Initialize type of tetrahedron
9:    $\rho = (\tau, e_\tau, e_{\kappa_1}, e_{\kappa_2}, t)$ 
10:  return  $\rho$ 
11: end function

```

---

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 in [13]. 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.

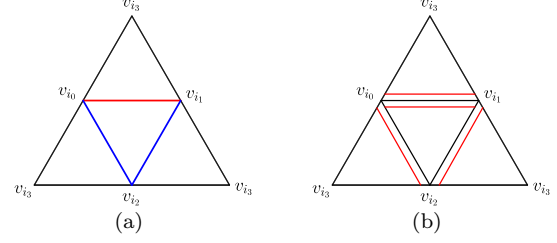
### 3.1 Marking edges: strict total order

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 straight-forward 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.

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.

### 3.2 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 5. The input of the function is a tetrahedron  $\tau = (v_0, v_1, v_2, v_3)$  and the output is the corresponding



**Figure 4:** Marked tetrahedra with: (a) our element-based marking method; and (b) the standard face-based marking method.

marked tetrahedron  $\rho$ . First, we obtain the consistent bisection edge,  $e_\tau$ , of the tetrahedron, see Line 2. Then, we obtain the opposite triangular faces of the vertices of the bisection edge  $e_\tau$ , see Lines 4–5. After that, we obtain the corresponding consistent bisection edges  $e_{\kappa_1}$  and  $e_{\kappa_2}$  of  $\kappa_1$  and  $\kappa_2$ , 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 5 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  $e_\tau$  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  $e_\tau$ ,  $e_{\kappa_1}$  and  $e_{\kappa_2}$  are connected generating the triangle  $(v_{i_0}, v_{i_1}, v_{i_2})$ , they define a planar configuration. Figure 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. 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(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(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

$$\begin{aligned} \kappa_1 &= (v_{i_0}, v_{i_1}, v_{i_2}), & \kappa_2 &= (v_{i_0}, v_{i_1}, v_{i_3}), \\ \kappa_3 &= (v_{i_0}, v_{i_2}, v_{i_3}), & \kappa_4 &= (v_{i_1}, v_{i_2}, v_{i_3}), \end{aligned}$$

are  $e_{\kappa_1} = (v_{i_0}, v_{i_1})$ ,  $e_{\kappa_2} = (v_{i_0}, v_{i_1})$ ,  $e_{\kappa_3} = (v_{i_0}, v_{i_2})$

---

**Algorithm 6** Conformingly marking a 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:      $\rho = \text{markTetrahedron}(\tau)$ 
5:      $\mathcal{T}' = \mathcal{T}' \cup \rho$ 
6:   end for
7:   return  $\mathcal{T}'$ 
8: end function

```

---

and  $e_{\kappa_4} = (v_{i_1}, v_{i_2})$ , respectively. Therefore, the refinement edge of  $\tau$  is  $e_\tau = e_{\kappa_1} = e_{\kappa_2}$  and the refinement faces are  $\kappa_1$  and  $\kappa_2$ . The faces  $\kappa_3$  and  $\kappa_4$  are the non-refinement faces and their marked edges are  $e_{\kappa_3}$  and  $e_{\kappa_4}$ , 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(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.

### 3.3 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 2.1. 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 5 leads to conformingly marked meshes.

Now, we can detail the method to conformingly mark an unstructured conformal tetrahedral mesh, see Algorithm 6. 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  $\rho$  for each tetrahedra  $\tau$  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.

## 4. RESTRICTED MARKED BISECTION

To bisect our unflagged planar meshes, we consider a restricted version of the standard marked bisection, see Algorithm 7. The restricted method bisects a tetra-

---

**Algorithm 7** Restrictd marked bisection.

---

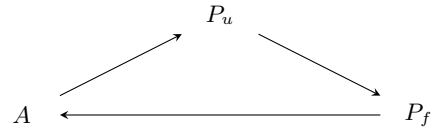
**input:** MarkedTetrahedron  $\rho$   
**output:** MarkedTetrahedron  $\rho_1$ , MarkedTetrahedron  $\rho_2$

```

1: function bisectTet( $\rho$ )
2:    $t = \text{type}(\rho)$ 
3:   if  $t$  is  $P_u$  then
4:      $\rho_1, \rho_2 = \text{bisectUnflaggedPlanar}(\rho)$ 
5:   else if  $t$  is  $P_f$  then
6:      $\rho_1, \rho_2 = \text{bisectFlaggedPlanar}(\rho)$ 
7:   else if  $t$  is  $A$  then
8:      $\rho_1, \rho_2 = \text{bisectAdjacent}(\rho)$ 
9:   end if
10:  return  $\rho_1, \rho_2$ 
11: end function

```

---

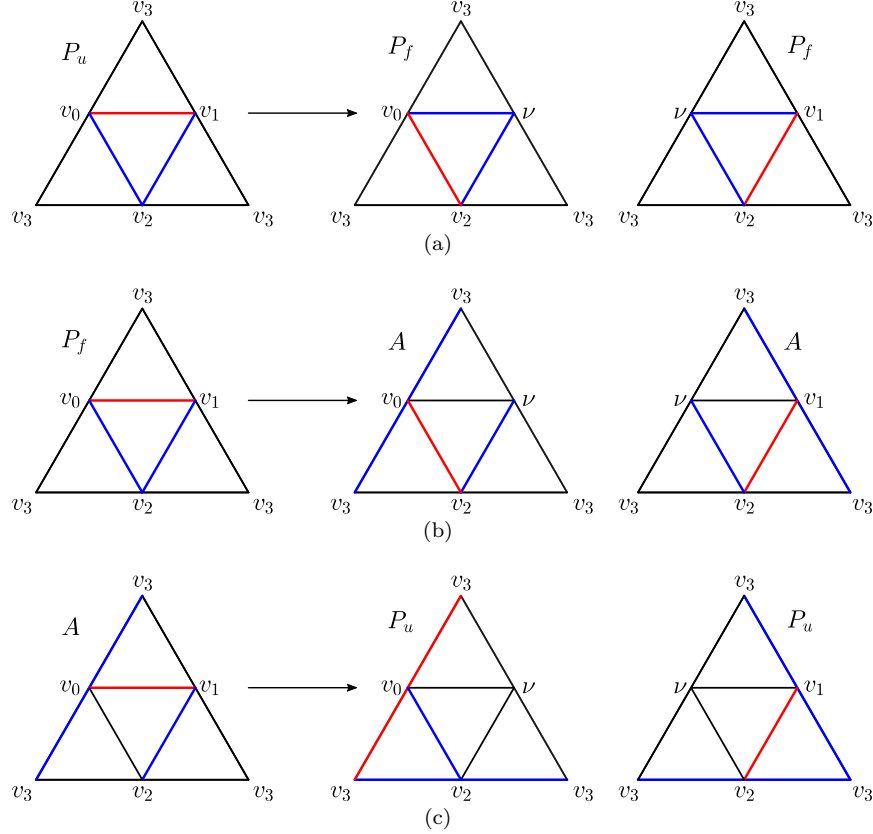


**Figure 5:** Restricted bisection cycle starting on unflagged planar type.

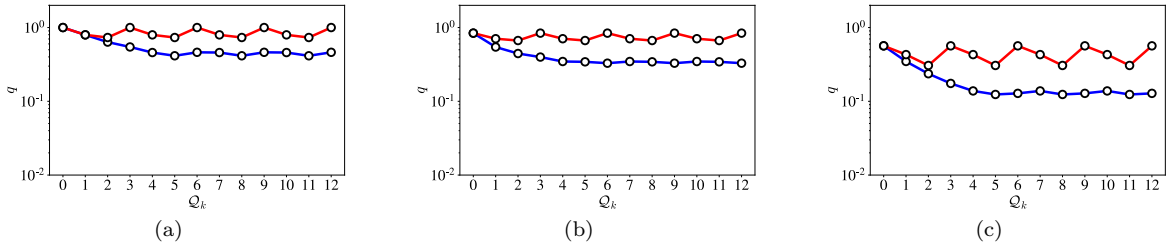
hedron according to its type. Moreover, it only needs to consider the bisection cycle of length three for the tetrahedron types  $P_u$ ,  $P_f$ , and  $A$ , see Figure 5. In the first case, Line 4, we bisect an unflagged planar tetrahedron. In the second case, Line 6, we bisect a flagged planar tetrahedron. Finally, in the third case, Line 8, we bisect an adjacent tetrahedron.

Figure 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  $\nu$  is the new vertex after the bisection of the edge  $e_\tau$ . 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 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  $P_u$ .



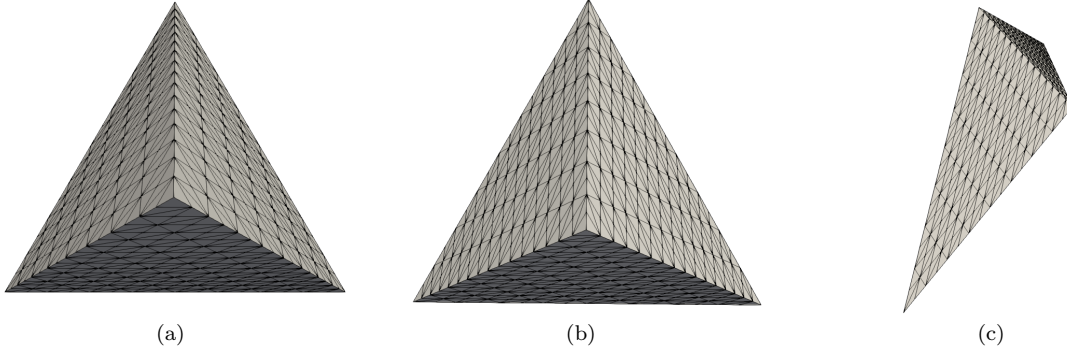
**Figure 7:** Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iteration: (a) equilateral tetrahedron; (b) cartesian tetrahedron; and (c) random tetrahedron.

## 5. 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 [20] of the mesh elements. 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 16GB. As a proof of concept, a mesh refiner has



**Figure 8:** Final mesh after twelve iterations of uniform refinement for: (a) equilateral tetrahedron; (b) cartesian tetrahedron; and (c) random tetrahedron.

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 work.

### 5.1 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,

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

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 7 plots the evolution of the minimum (blue line) and maximum (red line) qualities during the uniform refinement process. Figures 7(a), 7(b), and 7(c) illus-

trate 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 7(a), for the most symmetric tetrahedron, how the minimum quality achieves its minimum at iteration five, and then it remains cycling. For the cartesian and perturbed tetrahedra, we can see in Figures 7(b) and 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 8(a), 8(b), and 8(c) correspond to the meshes  $\mathcal{Q}_{12}$  of the equilateral, cartesian and random tetrahedra after 12 uniform refinements.

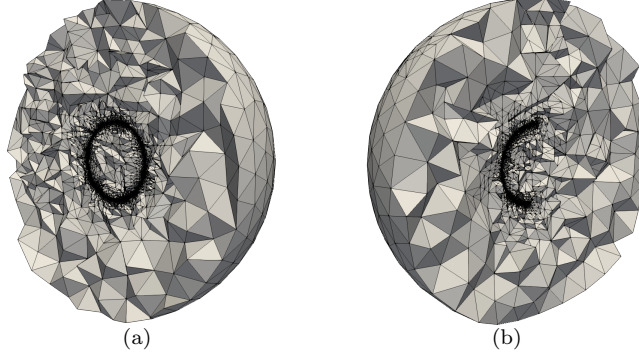
We have generated all the similarity classes for the three tetrahedra, and thus, the minimum mesh quality is achieved. Thus, this example illustrates that the method is stable and the mesh quality does not degenerate during successive refinement.

### 5.2 3D unstructured mesh: locally refining a sphere

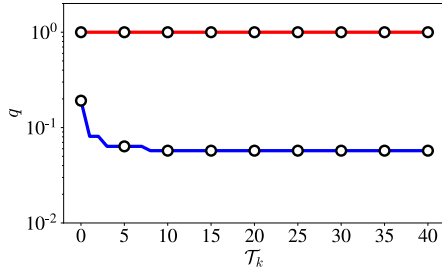
This example shows that the proposed refinement scheme can be applied to locally refine unstructured tetrahedral meshes. We recreate the first example from Maubach [8] and Arnold *et al.* [13] but for a sphere. Specifically, we generate an unstructured three-dimensional mesh of a sphere of radius 2 and centered at the origin. Let  $H$  be a hemisphere of a sphere of radius one centered at  $(1/2, 1/2, 1/2)$  defined by the equations

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

We want to adapt the tetrahedral mesh  $\mathcal{T}_0$  to the hemisphere  $H$ . At each local refine iteration, we choose



**Figure 9:** Slice of the mesh  $\mathcal{T}_{40}$  with the plane: (a)  $x = 1/2$ ; and (b)  $y = 1/2$ .



**Figure 10:** Quality of Example 5.2: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iterations.

the tetrahedra that intersect the hemisphere  $H$  as the refinement set. After forty iterations the mesh  $\mathcal{T}_{40}$  is composed by 5806615 tetrahedra and 1045175 vertices. Figures 9(a) and 9(b) show the  $\mathcal{T}_{40}$  sliced with the planes  $x = 1/2$  and  $y = 1/2$ . Figure 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.

### 5.3 3D space-time mesh: locally refining a iso-potential surface

The main goal of this example is to capture a three-dimensional manifold defined by the movement of a two-dimensional object. We show the evolution of the gravitational potential defined by two particles that

move along the  $y$ -axis. Let

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

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

the equation that defines the gravitational potential. For a given iso-value  $V_0$ ,  $V(\mathbf{x}, t) = V_0$  defines a two-dimensional embedded manifold in three-dimensional space. Let  $H$  be the hyper-cylinder with spherical basis defined by the equations

$$\left(x - \frac{1}{2}\right)^2 + \left(y - \frac{1}{2}\right)^2 = 1, \quad 0 \leq t \leq 1.$$

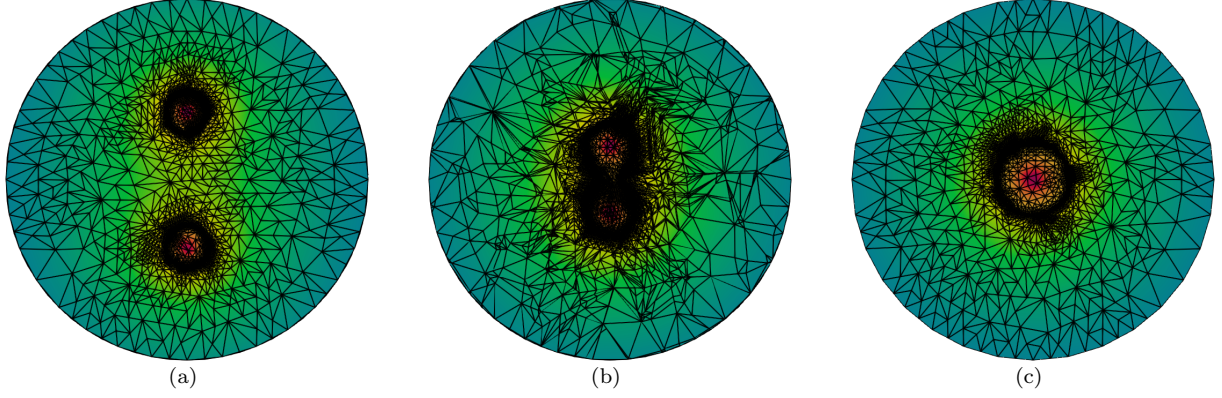
In this example, we choose the iso-value  $V_0 = -10$  and the parameters  $G = 1$ ,  $m_1 = 1$ ,  $m_2 = 1$ ,  $\mathbf{p}_1 = (1/2, 1/8)$ ,  $\mathbf{p}_2 = (1/2, 7/8)$  and  $v = 3/8$ .

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 vertices. We generate the set of tetrahedra that intersect  $H$ ,  $F_k = \{\tau \in \mathcal{T}_{k-1} \mid \sigma \cap H \neq \emptyset\}$ . Then, for each tetrahedron in  $F_k$  we compute the curvature of  $V(\mathbf{x}, t)$  at each simplex using the formula

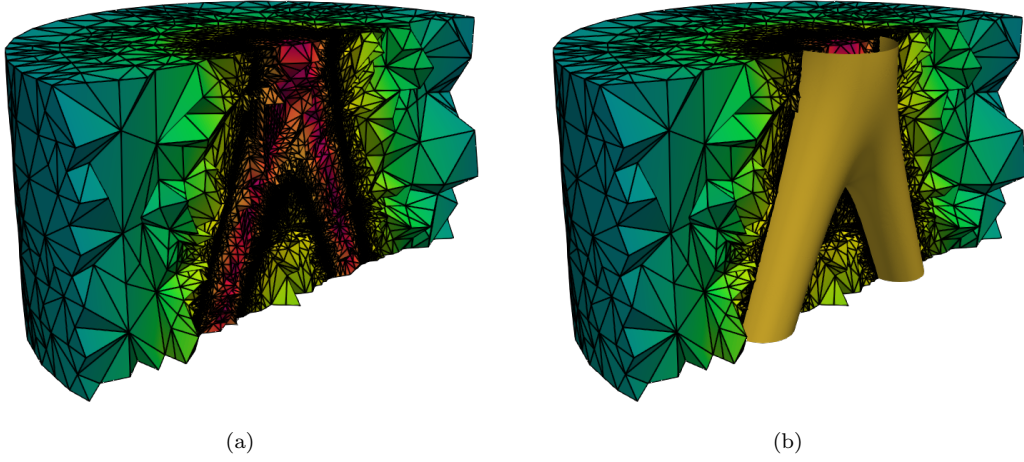
$$e_\sigma = \sum_{i=0}^3 \left| h_i^T \nabla^2 V(\mathbf{x}_i, t_i) h_i \right|,$$

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

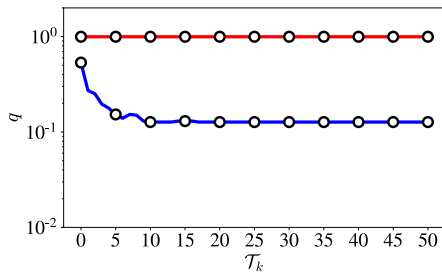
After 50 iterations of the local refinement process, the generated mesh  $\mathcal{T}_{50}$  has 8356894 tetrahedra and



**Figure 11:** Slices of  $\mathcal{T}_{50}$  with the plane: (a)  $t = 0.0$ ; (b)  $t = 0.5$ ; and (c)  $t = 1.0$ .



**Figure 12:** Slice of the  $\mathcal{T}_{50}$  with the plane  $x = 1/2$ , (a) with, and (b) without the iso-surface.



**Figure 13:** Quality of Example 5.3: Evolution of the maximum (red line) and minimum (blue line) mesh quality through the mesh refinement iterations.

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 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 12(b) shows the iso-surface that is extracted from the space-time mesh. Figure 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.

1504344 vertices. Figures 11(a), 11(b) and 11(c) show

## 6. 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.

## 7. ACKNOWLEDGMENTS

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] Rivara M.C. "Algorithms for refining triangular grids suitable for adaptive and multigrid techniques." *International journal for numerical methods in Engineering*, vol. 20, no. 4, 745–756, 1984
- [2] Rivara M.C. "Local modification of meshes for adaptive and/or multigrid finite-element methods." *Journal of Computational and Applied Mathematics*, vol. 36, no. 1, 79–89, 1991
- [3] Plaza A., Carey G.F. "Local refinement of simplicial grids based on the skeleton." *Applied Numerical Mathematics*, vol. 32, no. 2, 195–218, 2000
- [4] Plaza A., Rivara M.C. "Mesh Refinement Based on the 8-Tetrahedra Longest-Edge Partition." *IMR*, pp. 67–78, 2003
- [5] Mitchell W.F. "Adaptive refinement for arbitrary finite-element spaces with hierarchical bases." *Journal of computational and applied mathematics*, vol. 36, no. 1, 65–78, 1991
- [6] Mitchell W.F. "30 years of newest vertex bisection." *AIP Conference Proceedings*, vol. 1738, p. 020011. AIP Publishing, 2016
- [7] Kossaczky I. "A recursive approach to local mesh refinement in two and three dimensions." *Journal of Computational and Applied Mathematics*, vol. 55, no. 3, 275–288, 1994
- [8] Maubach J.M. "Local bisection refinement for n-simplicial grids generated by reflection." *SIAM Journal on Scientific Computing*, vol. 16, no. 1, 210–227, 1995
- [9] Maubach J.M. "The efficient location of neighbors for locally refined n-simplicial grids." *5th Int. Meshing Roundtable*, vol. 4, no. 6, 14, 1996
- [10] Traxler C.T. "An algorithm for adaptive mesh refinement in  $n$  dimensions." *Computing*, vol. 59, no. 2, 115–137, 1997
- [11] Stevenson R. "The completion of locally refined simplicial partitions created by bisection." *Mathematics of computation*, vol. 77, no. 261, 227–241, 2008
- [12] Alkämper M., Gaspoz F., Klöfkor R. "A Weak Compatibility Condition for Newest Vertex Bisection in Any Dimension." *SIAM Journal on Scientific Computing*, vol. 40, no. 6, A3853–A3872, 2018
- [13] Arnold D.N., Mukherjee A., Pouly L. "Locally adapted tetrahedral meshes using bisection." *SIAM Journal on Scientific Computing*, vol. 22, no. 2, 431–448, 2000
- [14] Coxeter H. "Discrete groups generated by reflections." *Ann. Math.*, pp. 588–621, 1934
- [15] Freudenthal H. "Simplizialzerlegungen von beschränkter flachheit." *Ann. Math.*, pp. 580–582, 1942
- [16] Kuhn H.W. "Some combinatorial lemmas in topology." *IBM Journal of research and development*, vol. 4, no. 5, 518–524, 1960
- [17] Bänsch E. "Local mesh refinement in 2 and 3 dimensions." *IMPACT of Computing in Science and Engineering*, vol. 3, no. 3, 181–191, 1991

- [18] Liu A., Joe B. “On the shape of tetrahedra from bisection.” *Mathematics of computation*, vol. 63, no. 207, 141–154, 1994
- [19] Liu A., Joe B. “Quality local refinement of tetrahedral meshes based on bisection.” *SIAM Journal on Scientific Computing*, vol. 16, no. 6, 1269–1291, 1995
- [20] Knupp P.M. “Algebraic mesh quality metrics.” *SIAM journal on scientific computing*, vol. 23, no. 1, 193–218, 2001



# PARALLEL FOUR-DIMENSIONAL ANISOTROPIC MESH ADAPTATION

Philip Claude Caplan

*Middlebury College Department of Computer Science, Middlebury, VT, U.S.A.,  
pcaplan@middlebury.edu*

## ABSTRACT

Anisotropic mesh adaptation is important for accurately predicting engineering quantities of interest with high-order finite element discretizations. Spacetime approaches, whereby time is treated as an additional spatial dimension and the mesh is fully coupled in space and time, have also shown to be effective in reducing the number of degrees of freedom needed to achieve a certain level of accuracy. Previous efforts have successfully demonstrated four-dimensional mesh adaptation capabilities so as to support adaptive simulations in  $3d + t$ . However, these algorithms have been restricted to sequential implementations. Parallel mesh adaptation strategies have become increasingly important in  $3d$  numerical simulations. Therefore, the goal of this work is to extend those strategies to perform parallel four-dimensional anisotropic mesh adaptation. We employ a functionality-first approach to parallelize an existing sequential mesh adaptation tool. This approach iteratively adapts the mesh while keeping partition interfaces fixed, and then migrates partition interfaces into the interior so they can be adapted during a subsequent pass within an adaptation iteration. We demonstrate that the algorithm scales well as the number of processors increases while maintaining good metric conformity for three- and four-dimensional problems.

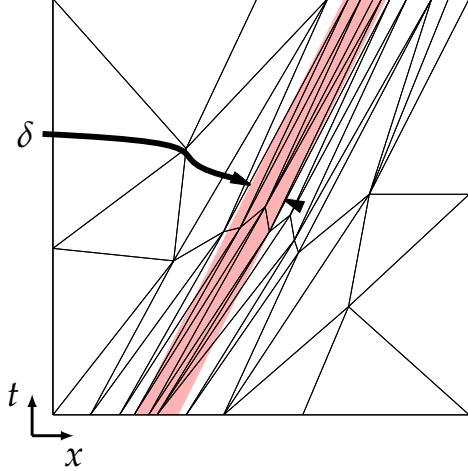
**Keywords:** mesh adaptation, anisotropic, parallel, four-dimensional, pentatope, spacetime

## 1. INTRODUCTION

Anisotropic mesh adaptation is fundamental for accurately predicting engineering quantities of interest with high-order finite element discretizations [1, 2]. To resolve unsteady physical phenomena in  $d$  dimensions, space and time can be coupled and the problem can be solved in a spacetime domain, whereby time is treated as an additional spatial dimension. This approach requires a  $(d + 1)$ -dimensional mesh, which means  $4d$  meshes are needed to solve unsteady  $3d$  problems.

Adaptation strategies in this spacetime domain can be categorized into (1) uniform, (2) tensor-product [3, 4, 5, 6, 7, 8] or (3) unstructured anisotropic approaches [9, 10, 11, 12, 13]. Yano [9] demonstrated that tensor-product approaches are effectively isotropic and, whereas they offer a substantial degree-of-freedom (DOF) savings over uniform refinement, they are dramatically outperformed by a fully un-

structured anisotropic approach. In the context of the one-dimensional problem of Figure 1 with a propagating feature of characteristic size  $\delta$ , Yano [9] experimentally observes that uniform, tensor-product, and unstructured anisotropic refinement approaches respectively require  $\mathcal{O}(\delta^{-2})$ ,  $\mathcal{O}(\delta^{-1})$  and  $\mathcal{O}(1)$  DOF to achieve the same level of accuracy in the simulation. This unstructured anisotropic spacetime approach has been extended to  $2d + t$  for convection-diffusion [9], wave equation [14] and oil reservoir simulations [11] and to  $3d + t$  for convection-diffusion problems [13]. Jayasinghe also demonstrates that a spatiotemporal approach scales very well with the number of parallel processors when compared to time-marching approaches [12]. However, to fully realize the potential of spacetime adaptive methods, large meshes, with millions or even billions of elements, are needed for practical engineering problems, thereby requiring a parallel mesh adaptation algorithm.



**Figure 1:** Unstructured spacetime mesh adaptivity can capture a feature with nominal size  $\delta$  using  $\mathcal{O}(1)$  elements, whereas tensor-product and uniform refinement methods would require  $\mathcal{O}(\delta^{-1})$  and  $\mathcal{O}(\delta^{-2})$  elements, respectively, to achieve the same level of accuracy.

Chrisochoides describes a telescopic approach for parallel mesh generation and adaptation at various levels of granularity [15]. Further, Tsolakis et al. give an excellent overview of the goals and strategies for parallel mesh adaptation [16]. The authors identify five criteria which can be used to evaluate parallel mesh adaptation codes: (1) stability, (2) reproducibility, (3) robustness, (4) scalability and (5) code reuse. Similar to the work of Tsolakis, we aim for stability and scalability. That is, we aim to maintain good metric conformity while ensuring the algorithm scales well as the number of processors is increased. Tsolakis also identifies two different approaches for parallel mesh generation and adaptation: functionality-first and scalability-first approaches. Codes such as EPIC [17] and FEFLO.A [18] fit into the functionality-first paradigm, whereby minimal changes are made to existing sequential mesh adaptation software to extend them to a parallel setting. Codes such as CDT3D [19] and REFINE [20] are classified as scalability-first approaches, whereby functionalities are completed as needed, but designed to be scalable across parallel processes first.

The primary goal of this work is to parallelize a four-dimensional mesh adaptation algorithm and demonstrate that good metric conformity is obtained as the number of processors increases. Our contribution is an extension of existing techniques for 3d parallel mesh adaptation to 4d, with a demonstration on benchmark cases. We also show the algorithm scales well with modest processor counts - we only present results for up to 36 processors due to limitations on our computa-

tional resources. We use a functionality-first approach whereby we maximize the code re-use of an existing sequential mesh adaptation algorithm. We first elaborate upon the sequential and parallel mesh adaptation strategies. The latter builds from the idea of fixing partition boundaries, and migrating the partition interfaces into the interior. We then study the algorithm on three- and four-dimensional problems and demonstrate that the parallel algorithm scales well as the number of processors increases, while also maintaining good metric conformity.

## 2. SEQUENTIAL MESH ADAPTATION STRATEGY

Our sequential mesh adaptation algorithm is based on the local cavity operator approach, initially described by Coupez [21, 22], later used by Loseille [23, 18] and extended to 4d by Caplan [13, 24, 25]. The inputs to the algorithm are (1) an initial mesh, (2) a metric field described at the vertices of the mesh and (3) the geometry entity associated with each vertex of the mesh. These entities could be geometry Nodes, Edges, Faces or Volumes (for a 4d geometry) which can be specified via a geometry engine, such as EGADS [26, 27] for 3d geometries. In our work, we study the geometry of a unit tesseract, which is bounded by 8 Cubes (Volumes), 10 Faces, 32 Edges and 16 Nodes. The topology of the tesseract is prescribed by traversing the vertex-facet incidence relations of each geometry entity [13]. Each vertex in the mesh is initially tagged with the lowest-dimensional geometry entity on which it lies, which is null for interior vertices that do not lie on the geometry.

The local cavity operator approach iteratively modifies the mesh  $\mathcal{T}^k$  by removing a set of cavity elements  $\mathcal{C}^k$  around a mesh facet  $f$  (such as a vertex, edge, triangle or tetrahedron), and then selecting a re-insertion vertex  $p$  to connect to the cavity boundary  $\partial\mathcal{C}^k$ , thereby creating new elements denoted by  $\mathcal{B}^k$ :

$$\mathcal{T}^{k+1} = \mathcal{T}^k \setminus \mathcal{C}^k(f) \cup \mathcal{B}^k(p, \partial\mathcal{C}^k). \quad (1)$$

The advantage of the cavity-based approach is that all classical mesh modification operators (splits, collapses, swaps and smoothing) can be reformulated in terms of Eq. 1 by the appropriate selection of  $f$  and  $p$ , thus simplifying the implementation in 4d.

To schedule the local mesh modification operators, a metric field is specified at each vertex of the initial mesh. That is, a symmetric, positive-definite  $d \times d$  tensor is prescribed at each vertex. The input mesh is retained in the background of the adaptation process - the metric at a new location in the mesh (either through an edge split or a vertex relocation) is ob-

tained by the log-Euclidean weighted average of the metrics from an element in this background mesh.

Following the conventions of the Unstructured Grid Adaptation Working Group (UGAWG) [16, 28, 29], the length of an edge between two vertices  $p$  and  $q$  is computed as

$$\ell_{\mathbf{m}}(\mathbf{p}, \mathbf{q}) \approx \ell_{\mathbf{m}(\mathbf{p})} \frac{r-1}{r \log r} \quad \text{with } r \equiv \frac{\ell_{\mathbf{m}(\mathbf{p})}}{\ell_{\mathbf{m}(\mathbf{q})}}, \quad (2)$$

where  $\ell_{\mathbf{m}(\mathbf{x})} = \sqrt{(\mathbf{q} - \mathbf{p})^t \mathbf{m}(\mathbf{x})(\mathbf{q} - \mathbf{p})}$ .

The quality of a  $d$ -simplex  $\kappa$  ( $d = 2$  for triangles, 3 for tetrahedra and 4 for pentatopes) is computed as

$$q_{\mathbf{m}}(\kappa) = \beta_n \frac{v_{\mathbf{m}}(\kappa)^{2/d}}{\sum_{e \in \mathcal{E}(\kappa)} \ell_{\mathbf{m}}^2(e)} \quad (3)$$

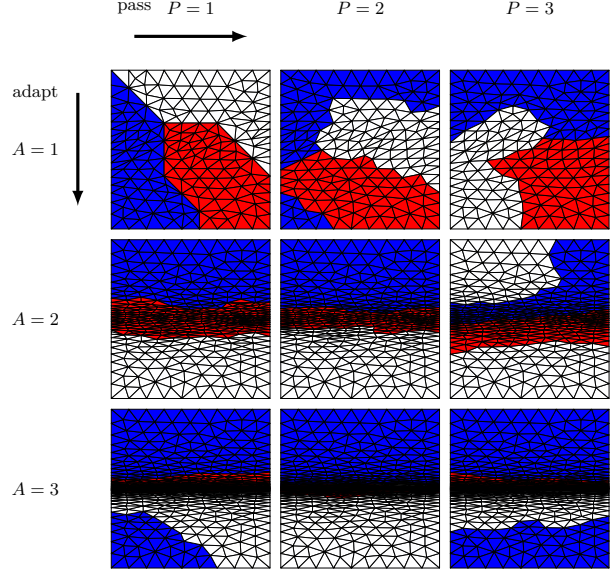
where  $\mathcal{E}$  is the set of  $d(d+1)/2$  edges of the simplex, and  $v_{\mathbf{m}}(\kappa)$  is the volume under the metric field:

$$v_{\mathbf{m}}(\kappa) \approx \sqrt{\det \mathbf{m}_{\nu}} v(\kappa), \quad \text{with } \nu = \arg \max_{\nu \in \kappa} \det \mathbf{m}_{\nu}. \quad (4)$$

The goal of the mesh adaptation algorithm is to produce a metric-conforming mesh in which all edges of the mesh have a length in the range  $[\sqrt{2}/2, \sqrt{2}]$  and all elements have a quality greater than 0.8. To achieve these goals, the algorithm begins by iteratively collapsing all edges shorter than  $\sqrt{2}/2$  and then performing edge splits on edges longer than 2 without introducing new short edges. Next, edge swaps are used to improve the quality of the elements in two passes: first all edges adjacent to an element with quality less than 0.4 are targeted, then all elements with a quality less than 0.8 are targeted. A local vertex smoothing procedure is then used to drive the edge lengths surrounding a particular vertex to 1. This overall procedure is then repeated using a target edge length of  $\sqrt{2}$  for the split operator. For further details on this algorithm, please see the work of Caplan [13, 25].

### 3. PARALLEL MESH ADAPTATION STRATEGY

We use a functionality-first approach to parallelize the mesh adaptation algorithm and attempt to re-use as much of the sequential mesh adaptation algorithm as possible. This approach is similar to the work of Dignonet [30, 31, 32] and Lachat [33, 34, 35, 36] in which the mesh is divided into *processing elements* to be remeshed by a third-party sequential remesher, such as MMG3D [37]. Other coarse-grained parallel implementations which modify partition interiors, while keeping interfaces fixed include REFINE [20],



**Figure 2:** Parallel adaptation strategy. Each row corresponds to a single adaptation iteration ( $A$ ), whereas each column corresponds to a pass ( $P$ ) within the adaptation iteration. Each partition (three in this example) is highlighted with a different color. The goal of the repartitioner after each pass is to migrate the interfaces between partitions into the partition interiors.

EPIC [17] and FEFLO.A [38, 39, 18]. CDT3D uses a fine-grained approach to modifying the mesh, designed for shared-memory parallel mesh adaptation [40].

The main idea behind the parallel mesh adaptation algorithm is to decompose the initial mesh (or receive a decomposed mesh from a numerical simulator) into partitions and adapt the interior of each partition while keeping the partition boundaries fixed. As long as the sequential mesh adaptation algorithm allows for these partition boundaries to be frozen, then the sequential algorithm can be used within each partition without requiring any intrusive changes to communicate boundary modifications. Communicating these boundary modifications would require special treatment for each mesh modification operator and would need to be directly incorporated into the sequential mesh adaptation code [41], which we chose to avoid. In our setting, however, the difficulty lies in the fact that partition boundaries must be migrated to the partition interiors so as to ensure the mesh is metric-conforming after each adaptation iteration. In order to migrate the partition boundaries into the interior, a series of “passes” are used within a particular iteration of the global mesh adaptation procedure - each pass is then followed by a repartition of the mesh. Here we use PARMETIS [42] to repartition the mesh, though other

tools such as PT-SCOTCH [43] or ZOLTAN [44] could be used. The simplex-to-simplex adjacencies are used to construct the graph that is passed to PARMETIS. That is, each simplex is a vertex of the graph, and an edge in the graph is created if two simplices share a common  $(d - 1)$ -facet (i.e. a  $(d - 1)$ -simplex). The primary goal of the repartitioning procedure is to migrate existing partition boundaries into the interior for subsequent passes. However, some  $(d - 1)$ -facets in the partition interfaces may not require modification, therefore, it would be less necessary to migrate these into the interior, when other areas in the mesh may require more attention. As a result, we penalize edges by imposing edge weights which are computed via the “age” of the vertices of the elements in the mesh, an idea inspired by REFINE [20]. The age of a vertex is defined as the number of rejected mesh modifications of any facet surrounding the vertex. Whenever a local cavity operator is accepted, the age of any vertices in the re-inserted ball become zero. Note that this procedure is heuristic, and does not *guarantee* that partition boundaries are migrated to interior.

An example of the parallel mesh adaptation strategy in  $2d$  is given in Fig. 2. Each row corresponds to a particular adaptation iteration ( $A$ ), whereas each column corresponds to a pass within the adaptation iteration ( $P$ ). To clarify, each adaptation iteration corresponds to an iteration of the adaptive numerical simulation, whereby a solution is obtained, the error is estimated and an optimal metric field is computed. This metric field, along with the current mesh, is then passed to the mesher. Within this call to the mesher, multiple *passes* are performed (with the same metric field) with the goal of migrating the partition interfaces so as to adapt any necessary regions of the domain before returning the next mesh to the solver.

In Fig. 2 we can see that the boundaries of the partitions are migrated to the partition interiors from one pass to the next - however, in some cases it is not possible to migrate *every* interface  $(d - 1)$ -facet. Also observe that partitions may be divided across multiple connected components, as in Fig. 2 for  $A = 2$  and  $P = 3$ . Furthermore, partition boundaries may exhibit arbitrarily shaped geometries, especially in  $3d$  and  $4d$ . The only modification required in our sequential mesh adaptation algorithm was to allow for vertices to be fixed during the adaptation. In particular, each partition fixes any vertex along a partition interface. We additionally needed to add checks for the non-manifold vertices that are produced for arbitrarily shaped partitions.

One difficulty is that we would like all global indices of the fixed vertices to remain the same across all processors. This is not a problem when an edge is split, swapped, or when a vertex is smoothed, however, spe-

cial care is required for the collapse operator. Since the collapse requires the removal of a vertex, any element that references a vertex with a higher index than the removed vertex must have its vertex indices decremented. This would require communication between processors whenever a collapse is performed. Therefore, all fixed vertices are initially moved to the front of the global list of vertices in the mesh. Other synchronization techniques are possible, but this was the simplest to implement within our framework.

After each re-partitioning procedure, the processors must communicate which elements are retained, which are sent to other partitions, and which are received from other processors. We use the Message Passing Interface (MPI) to exchange this information between processors [45]. The communication cost of this exchange operation can be expensive, which we study in the following sections.

Similar to the example in Fig. 2 we use three passes per adaptation iteration since we experimentally observed that this achieved good metric conformity while not being too costly.

## 4. NUMERICAL EXPERIMENTS

To evaluate the parallel mesh adaptation algorithm, we follow an approach similar to the work of Tsolakis [16, 46]. Starting with an initial mesh, conforming to a prescribed metric, we scale the metric and call our parallel mesh adaptation algorithm using a prescribed number of processors. The difference between our work and the work of Tsolakis, however, is that since our sequential mesh adaptation algorithm is designed to make small changes in a mesh, we scale the metric iteratively instead of scaling it to a desired complexity all at once. Specifically, our sequential mesh adaptation algorithm works best when the incoming edge lengths are between  $[0.5, 2.0]$ , motivated by the MOESS algorithm [9, 47, 48].

We study two types of cases: (1) when the metric field is analytic and (2) when the metric field is obtained from the MOESS algorithm, and only available at the discrete vertices of the initial mesh. When the metric field is analytic, each iteration  $i$  of our scaling procedure consists of evaluating the prescribed metric on the current mesh and then scaling the metric by a factor  $s = \sqrt{2}^i$ . When the metric field is discrete, we simply use a scaling factor of  $s = \sqrt{2}$  on the metric of the *current* mesh in the scaling iteration. Each component of the metric field is multiplied by the scaling factor  $s$ , which amounts to a multiplication of each eigenvalue of the metric field by  $s$ . Thus, the number of elements increases by a factor of  $\sqrt{s}^d$  after each iteration, while maintaining the same anisotropic ratios as in the initial mesh.

**scaleMeshParallel**

```

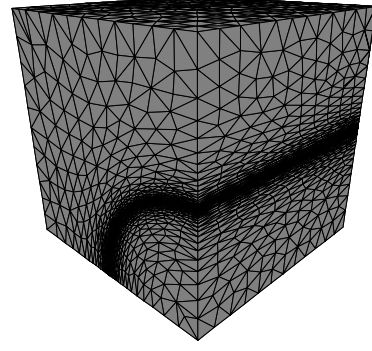
input: initial mesh  $\mathcal{M}$ , metric  $\mathbf{m}$ 
output:  $\mathcal{M}$ 
1  $\mathcal{M}_p \leftarrow \text{partition}(\mathcal{M}, p)$ 
2 for  $i = 1$  to  $n_{\text{iter}}$ 
3    $\mathbf{m}_p = \text{evaluateMetric}(\mathbf{m}, \mathcal{M}_p)$ 
4   fix partition boundary  $\partial\mathcal{M}_p$ 
5   for  $j = 1$  to  $n_{\text{pass}}$ 
6      $\mathcal{M}_p \leftarrow \text{adapt}(\mathcal{M}_p, \mathbf{m}_p)$ 
7     balance the mesh (repartition & exchange)
8 accumulate meshes from all processors into  $\mathcal{M}$ 

```

**Algorithm 1:** Scaling an initial mesh  $\mathcal{M}$  according to a metric field  $\mathbf{m}$  (either discrete or analytic) in parallel. The input mesh is initially partitioned and distributed across all parallel processors. For each adaptation iteration, the metric is re-evaluated on the current mesh and scaled to produce the next mesh in the adaptation sequence. Each adaptation iteration involves  $n_{\text{pass}}$  passes of the parallel adaptation algorithm so as to migrate partition interfaces to the interior. The final mesh is then accumulated on the root processor in order to evaluate metric conformity.

This procedure is outlined in Algorithm 1. After performing an initial partitioning of the mesh (line 1), each processor receives a subdomain mesh  $\mathcal{M}_p$ . We then iterate until the desired complexity is reached. Each processor evaluates the metric on the current mesh and scales it according to the scaling iteration counter. Then the parallel mesh adaptation procedure begins by computing the partition boundaries, labelling vertices which should be fixed, and then moving fixed vertices to the beginning of the vertex list (line 4). Each processor then performs 3 passes of adapting the mesh, migrating partition interfaces to the interior (with a penalty on the age of an element), and then exchanging the elements (and discrete metrics at the vertices) with all other processors. When the procedure is complete, all meshes are accumulated into the final mesh  $\mathcal{M}$  (line 8) which is then used to evaluate metric conformity. The latter is simply needed as a post-processing procedure which does not contribute to the analysis of the timing results. In the following, we measure the time spent adapting the mesh (line 6) and performing the load balancing, which consists of repartitioning and exchanging elements across processors (line 7) since these are the most costly operations. Adapting the mesh is clearly the bottleneck we wish to parallelize, but a poor design of the mesh exchange data structures and procedure can cause a significant overhead that is counterproductive to the parallel mesh adaptation process.

In each of the following subsections, we will first assess the ability of the parallel implementation to conform to the scaled metric field, whether analytic or discrete. We will then revisit the scalability of the algorithm in



**Figure 3:** Initial 61k tetrahedra mesh for the Cube-Polar case.

the final subsection.

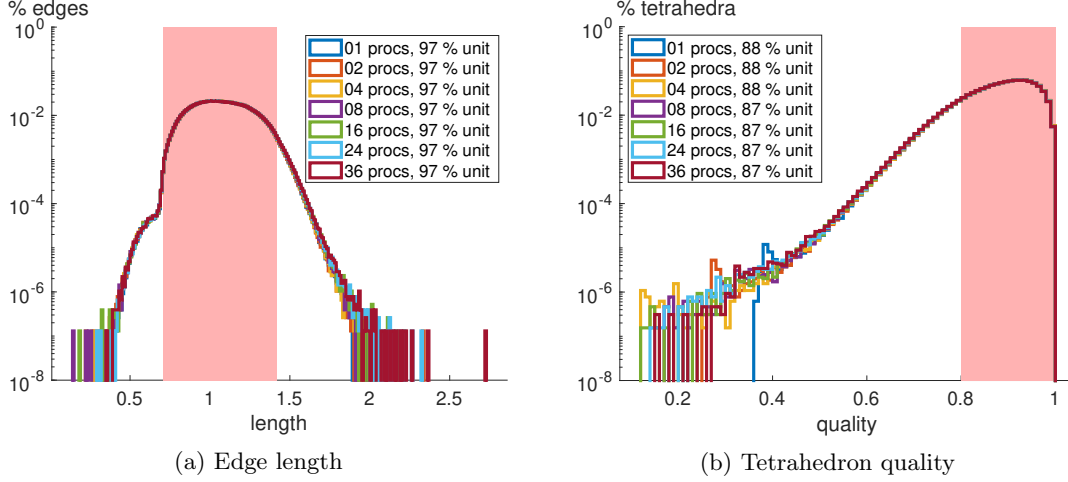
#### 4.1 Cube Polar (CP)

Before delving into some 4d cases, let us first evaluate the algorithm on a 3d benchmark case proposed by the UGAWG. The initial mesh passed into Alg. 1 is generated by adapting an initially structured mesh to the analytic metric:

$$\mathbf{m}(\mathbf{x}) = \mathbf{q} \text{diag}(h_r^{-2}, h_\theta^{-2}, h_z^{-2}) \mathbf{q}^t \quad (5)$$

where  $h_z = 0.1$ ,  $h_r = 0.001 + 0.198|r - 0.5|$  and  $h_\theta = 0.1d + 0.025(1 - d)$  where  $d = \min(10|r - 0.5|, 1)$ . Note that  $r = \sqrt{x^2 + y^2}$  and  $\theta = \arctan(y, x)$ . The eigenvectors  $\mathbf{q}$  consists of the unit vectors representing the cylindrical system unit vectors in a Cartesian system. The initial mesh, consisting of 63,237 tetrahedra and 12,242 vertices, is shown in Fig. 3. Alg. 1 is then used to scale this initial mesh and analytic metric to a mesh with approximately 6.5 million tetrahedra using various numbers of processors. Specifically, Alg. 1 is run with 1, 2, 4, 8, 16, 24 and 36 processors. The edge length and quality histograms of Fig. 4 demonstrate that the final meshes exhibit excellent metric conformity, regardless of the number of processors that were used to scale the mesh. Table 2 further shows that about 97% of the edges are in the quasi-unit range and the number of tetrahedra with a quality greater than 0.8 is at least 87% for all cases. Furthermore, the number of elements in the final mesh is almost identical across each test case. Although this test case is still relatively small in the context of a practical engineering simulation, it is large enough such that the processor interface elements can be effectively migrated to the interior so as to achieve good metric conformity.





**Figure 4:** Edge length and pentatope quality distributions for the 3d Cube-Polar case.

**Table 1:** Metric conformity for the Cube-Polar case running with  $p$  processors, generating  $n_{\text{elem}}$  tetrahedra: min. ( $\ell_{\min}$ ) and max. ( $\ell_{\max}$ ) edge lengths, percentage of edges in the quasi-unit range ( $\% \ell_{\text{unit}}$ ), min. quality ( $q_{\min}$ ) and percentage of tetrahedra with quality  $\geq 0.8$  ( $\% q_{\text{unit}}$ ).

$p$	$\ell_{\min}$	$\ell_{\max}$	$\% \ell_{\text{unit}}$	$q_{\min}$	$\% q_{\text{unit}}$	$n_{\text{elem}}$
1	0.41	2.11	97%	0.36	88%	6.46m
2	0.28	2.33	97%	0.25	88%	6.46m
4	0.22	2.10	97%	0.12	88%	6.45m
8	0.14	2.19	97%	0.17	87%	6.45m
16	0.18	2.21	97%	0.12	87%	6.45m
24	0.29	2.33	97%	0.14	87%	6.45m
36	0.36	2.73	97%	0.15	87%	6.45m

**Table 2:** Metric conformity for the Tesseract-Linear case running with  $p$  processors, generating  $n_{\text{elem}}$  pentatopes: min. ( $\ell_{\min}$ ) and max. ( $\ell_{\max}$ ) edge lengths, percentage of edges in the quasi-unit range ( $\% \ell_{\text{unit}}$ ), min. quality ( $q_{\min}$ ) and percentage of pentatopes with quality  $\geq 0.8$  ( $\% q_{\text{unit}}$ ).

$p$	$\ell_{\min}$	$\ell_{\max}$	$\% \ell_{\text{unit}}$	$q_{\min}$	$\% q_{\text{unit}}$	$n_{\text{elem}}$
1	0.16	2.43	93%	0.25	52%	7.47m
2	0.23	2.53	93%	0.29	51%	7.42m
4	0.20	3.00	93%	0.27	51%	7.37m
8	0.16	4.60	92%	0.20	50%	7.32m
16	0.08	4.88	89%	0.11	45%	6.66m
24	0.11	5.37	86%	0.11	41%	5.95m
36	0.07	4.55	84%	0.07	37%	5.63m

## 4.2 Tesseract Linear (TL)

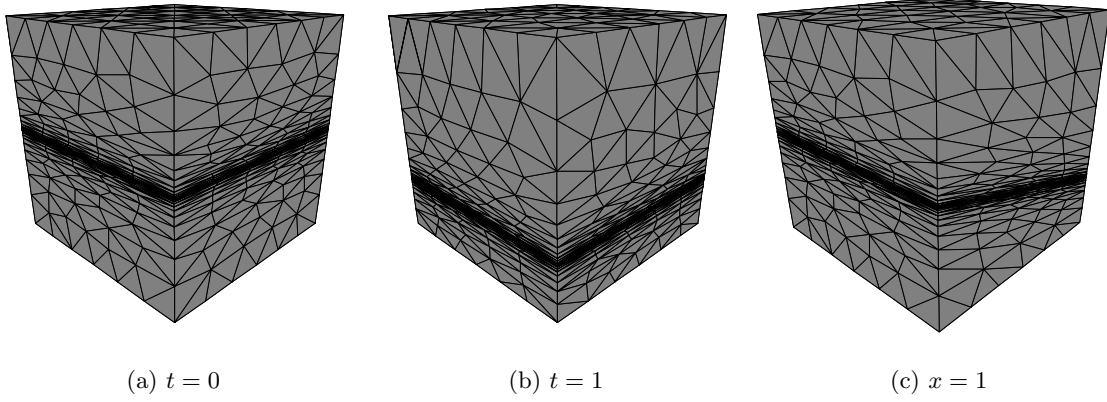
Now, we will study some 4d problems. The initial mesh for the Tesseract Linear (TL) case was generated from our sequential 4d mesh adaptation code using an analytic metric that emulates a moving boundary layer, described by

$$\mathbf{m}(\mathbf{x}) = \mathbf{q} \text{diag}(h_x^{-2}, h_y^{-2}, h_z^{-2}, h_t^{-2}) \mathbf{q}^t$$

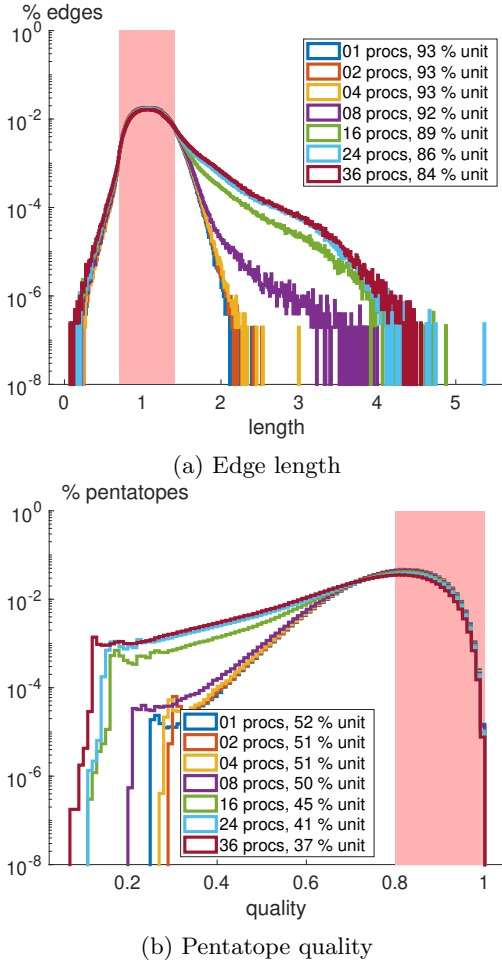
where  $\mathbf{q}$  is a rotation matrix of an angle  $\alpha = \arctan(0.25, 1)$  radians about the  $xy$  plane,  $h_x = h_y = h_z = 0.2$  and  $h_z = 0.002 + 0.396d$  with  $d = |\cos \alpha z - \sin \alpha t - 0.5 \cos \alpha|$ , which is the distance to the rotated plane of the boundary layer. The initial mesh contains 120,054 pentatopes and 8,297 vertices. The boundaries of this initial mesh, extracted at the  $t = 0$ ,  $t = 1$  and  $x = 0$  hyperplanes are shown in Fig. 5. Using Alg. 1 with this initial mesh, the analytic metric is scaled to preserve the anisotropy ratios to eventually produce meshes with 6-7 million pentatopes. Fig. 6

shows the length and quality histograms, measured under the scaled analytic metric, for the final meshes produced using 1, 2, 4, 8, 16, 24 and 36 processors. Overall, metric conformity is quite similar for all numbers of processors, with a slight degradation in the fraction of quasi-unit edges and quasi-unit pentatopes as the number of processors increases. The final meshes are still quite small - the added number of processors for a small mesh produces more interface elements that are frozen during the parallel adaptation process. These metric conformity results are summarized in Table 2. With 1–8 processors, over 90% of the edges are in the quasi-unit range, and over 50% of the pentatopes have a quality above 0.8. These results are similar to initial ratios of the edges and pentatopes within the quasi-unit ranges for the initial mesh of 120k pentatopes. These statistics, however, degrade to 89%, 86% and 84% in the fraction of quasi-unit edges, and 45%, 41% and 37% in quasi-unit pentatopes for 16, 24 and 36 processors, respectively.





**Figure 5:** Boundaries of the initial 100k pentatope mesh for the Tesseract-Linear case.



**Figure 6:** Edge length and pentatope quality distributions for the 4d Tesseract Linear case.

### 4.3 Tesseract Wave (TW)

Next, we consider a case in which the initial mesh was obtained from the Mesh Optimization via Error Sampling and Synthesis (MOESS) algorithm. Specifically, the mesh was obtained by adapting to the  $L^2$  error in the function

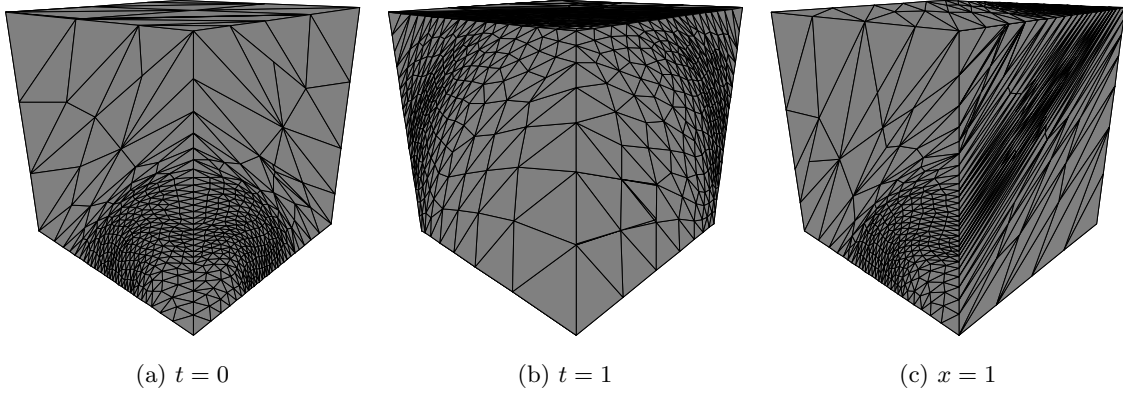
$$u(\mathbf{x}, t) = \exp(-t) \exp(-200(r(t) - \|\mathbf{x}\|)^2),$$

with  $r(t) = 0.4 + 0.7t$ , using a linear discontinuous Galerkin discretization. The initial mesh consists of 233,248 pentatopes and 14,515 vertices - the boundaries of this mesh, extracted at the  $t = 0$ ,  $t = 1$  and  $x = 0$  hyperplanes are shown in Fig. 7. Due to symmetry, we are only modeling one eighth of the expanding spherical wave. The sphere at it's initial radius is visible at  $t = 0$  (Fig. 7a), and at it's final radius at  $t = 1$  (Fig. 7b). Along hyperplanes with a non-constant  $t$ , we should see the geometry of a 3d cone, which is the projection of the expanding sphere in 4d (a 4d hypercone). The final metric obtained from the MOESS algorithm is then saved and used as the initial discrete metric, which is then scaled according to Alg. 1 to achieve meshes with approximately 7 million pentatopes.

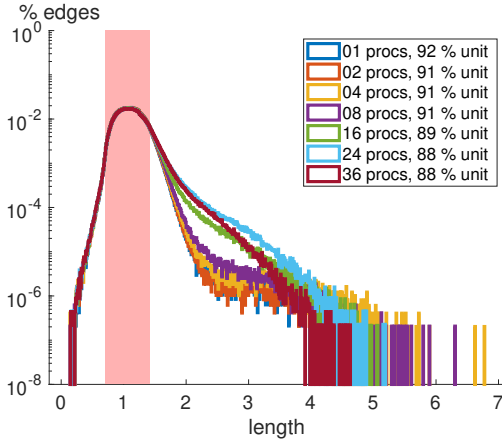
As in the previous case, metric conformity is excellent (90% of the edges have a length in the quasi-unit range, and 40% of the pentatopes have a quality greater than 40%) when the number of processors is still quite low - specifically up to 8 processors. The edge length and pentatope quality histograms are shown in Fig. 8 and the metric conformity statistics are summarized in Fig. 3. Again, metric conformity slightly degrades when the mesh is over decomposed.

### 4.4 Scalability

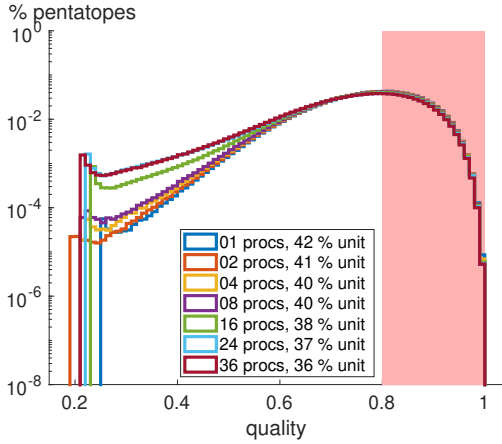
Finally, we analyze the scalability of our parallel mesh adaptation implementation. Due to limitations on our



**Figure 7:** Boundaries of the initial 200k pentatope mesh for the Tesseract-Wave case.



(a) Edge length



(b) Pentatope quality

**Figure 8:** Edge length and pentatope quality distributions for the 4d Tesseract Wave case.

**Table 3:** Metric conformity for the Tesseract-Wave case running with  $p$  processors, generating  $n_{\text{elem}}$  pentatopes: min. ( $\ell_{\min}$ ) and max. ( $\ell_{\max}$ ) edge lengths, percentage of edges in the quasi-unit range ( $\% \ell_{\text{unit}}$ ), min. quality ( $q_{\min}$ ) and percentage of pentatopes with quality  $\geq 0.8$  ( $\% q_{\text{unit}}$ ).

$p$	$\ell_{\min}$	$\ell_{\max}$	$\% \ell_{\text{unit}}$	$q_{\min}$	$\% q_{\text{unit}}$	$n_{\text{elem}}$
1	0.20	4.82	92%	0.25	42%	6.86m
2	0.20	5.37	91%	0.19	41%	6.99m
4	0.14	6.77	91%	0.21	40%	7.05m
8	0.14	6.31	91%	0.21	40%	6.90m
16	0.18	5.06	89%	0.23	38%	6.59m
24	0.17	5.19	88%	0.22	37%	6.19m
36	0.15	4.63	88%	0.21	36%	6.37m

computational resources, this study is restricted to smaller numbers of processors (up to 36). For runs with 1 – 24 processors, we used **node A** of Table 4 and we used **node B** for runs with 36 processors. For every test case, we report the total time to scale the initial mesh to the final mesh (i.e. the total time spent in Alg. 1). We also break down this total cost into the time spent adapting and balancing the mesh, since these were the most costly. The remaining time was spent synchronizing the indices, pre-processing the partitions (including moving the fixed vertices to the front of the arrays), and the actual re-partitioning of the mesh using PARMETIS.

The timing results are tabulated in Tables 5, 6 and 7. The total number of iterations used in Alg. 1 was 7, 6 and 10 for the Tesseract Linear, Tesseract Wave and Cube Polar cases, respectively. We also report the number of elements  $n_{\text{elem}}$  and partition interface  $(d - 1)$ -facets  $n_{\partial p}$ . Note that, in the case of the sequential algorithm ( $p = 1$ ), we still perform the algorithm with 3 passes, but only report the timing for a single pass.

**Table 4:** Machines used in the scalability analysis.

<b>node A</b>	28 × Intel Xeon Gold 5120 @ 2.20GHz
<b>node B</b>	36 × Intel Xeon Gold 6140 @ 2.30GHz

Next, we analyze the scalability of the parallel algorithm with respect to the sequential version. To be fair to the sequential adaptation algorithm, we only include adaptation time for one pass in our scalability analysis - there is clearly no need to include the time to migrate and re-partition the mesh, but we still perform some of the pre-processing steps which is ignored in the timing of the sequential algorithm. Thus, the final speedup of the parallel algorithm is measured as the total time (for all passes of the parallel algorithm) divided by the time for one pass of the sequential algorithm (accumulated across all iterations), which is further normalized by the number of elements since each test case produced a different number of elements. The speedup obtained with various processors for all three test cases considered in this paper is plotted in Fig. 9. A dashed line is used to show the ideal linear speedup. Our algorithm exhibits a linear speedup with only a few processors (up to 8), however, it begins to exhibit a superlinear speedup with more processors.

**Table 5:** Total time ( $t_{\text{total}}$ ) for the Cube-Polar case running with  $p$  processors, generating  $n_{\text{elem}}$  tetrahedra, broken into adaptation (% adapt) and interface migration (% bal.) times, with the number of interface triangles ( $n_{\partial P}$ ) in the final repartitioned mesh.

$p$	$n_{\text{elem}}$	$n_{\partial P}$	$t_{\text{total}}$	% adapt	% bal.
1	6.46m	-	1d:4h	100.0%	-
2	6.46m	20.0k	1d:24m	93.1%	6.3%
4	6.45m	39.3k	7h:9m	90.1%	8.9%
8	6.45m	72.4k	2h:33m	87.8%	10.7%
16	6.45m	118.3k	56m:32s	86.7%	11.5%
24	6.45m	144.5k	31m:11s	86.2%	11.8%
36	6.45m	178.5k	16m:37s	82.6%	15.5%

The work performed by the mesh adaptation algorithm is not linear in the size of the mesh, so the speedup will be superlinear when the problem size is evenly distributed across parallel processors. A previous analysis of the sequential mesh adaptation operators [25] suggests the work performed by, for example, the insertion operator increases linearly with the size of the mesh. Thus, the total work required to insert vertices for a particular problem is roughly the number of edges to be split times the size of the mesh. In parallel, the number of edges to be split is ideally distributed evenly across all processors, and the size of the partition is clearly smaller than the total size of the mesh. Thus the cost of the insertion operator is dramatically reduced (more than linear) when the work is divided across a large number of parallel pro-

**Table 6:** Total time ( $t_{\text{total}}$ ) for the Tesseract-Linear case running with  $p$  processors, generating  $n_{\text{elem}}$  pentatopes, broken into adaptation (% adapt) and interface migration (% bal.) times, with the number of interface tetrahedra ( $n_{\partial P}$ ) in the final repartitioned mesh.

$p$	$n_{\text{elem}}$	$n_{\partial P}$	$t_{\text{total}}$	% adapt	% bal.
1	7.25m	-	2d:6h	100.0%	-
2	7.42m	57.7k	2d:1h	98.2%	1.4%
4	7.37m	111.0k	15h:57m	97.7%	1.7%
8	7.32m	193.2k	7h:35m	97.4%	1.9%
16	6.66m	129.0k	2h:24m	97.6%	1.7%
24	5.95m	69.8k	1h:21m	98.0%	1.3%
36	5.63m	285.7k	42m:11s	97.2%	2.1%

**Table 7:** Total time ( $t_{\text{total}}$ ) for the Tesseract-Wave case running with  $p$  processors, generating  $n_{\text{elem}}$  pentatopes, broken into adaptation (% adapt) and interface migration (% bal.) times, with the number of interface tetrahedra ( $n_{\partial P}$ ) in the final repartitioned mesh.

$p$	$n_{\text{elem}}$	$n_{\partial P}$	$t_{\text{total}}$	% adapt	% bal.
1	6.72m	-	2d:17h	100.0%	-
2	6.99m	100.7k	3d:1h	98.4%	1.3%
4	7.05m	155.2k	1d:2h	98.3%	1.3%
8	6.90m	191.3k	8h:19m	97.9%	1.6%
16	6.59m	214.7k	2h:59m	97.7%	1.7%
24	6.19m	184.0k	1h:21m	97.5%	1.9%
36	6.37m	259.9k	53m:50s	96.7%	2.6%

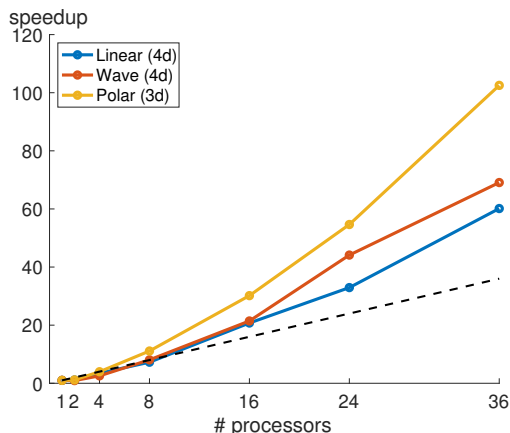
cessors. The fact that the mesh adaptation algorithm exhibits this superlinear convergence while maintaining reasonable metric conformity is a promising result for large-scale  $3d + t$  numerical simulations. Finally, we report the time (and number of adaptation iterations) to scale the mesh to much larger sizes with 36 processors for the same three test cases. All results were obtained on **node B** of Table 4. Total run time is reasonable, taking less than two days to perform 10 adaptation iterations to achieve a mesh with 69 million pentatopes, and roughly 9 hours to perform 14 adaptation iterations to achieve a mesh with about 52 million tetrahedra. Furthermore, metric conformity is better with these large meshes (compared with the results on smaller meshes in the previous section).

## 5. CONCLUSIONS

We have demonstrated the implementation of a parallel four-dimensional mesh adaptation for  $3d + t$  space-time numerical simulations. The developed algorithm builds off existing  $3d$  mesh adaptation algorithms in which the mesh adaptation problem is divided across a specific number of processors. Within a particular pass of an adaptation iteration, the boundaries of each

**Table 8:** Total time to adapt the mesh to 52 (Polar), 69 (Linear) and 47 (Wave) million simplices, along with metric conformity statistics on quasi-unit edge lengths and simplex quality.

Case	iter.	$n_{\text{elem}}$	$n_{\partial P}$	$t_{\text{total}}$	% adapt	% bal.	% $\ell_{\text{unit}}$	% $q_{\text{unit}}$
Polar (3d)	14	51.70m	703.7k	8h:59m	70.8%	28.1%	97.0%	88.0%
Linear (4d)	10	69.00m	1.779m	1d:17h	97.7%	1.7%	87.0%	45.0%
Wave (4d)	9	47.40m	1.173m	17h:24m	97.4%	2.2%	90.0%	45.0%



**Figure 9:** Parallel speedup when adapting the meshes for the Cube-Polar (Polar), Tesseract Linear (Linear) and Tesseract Wave (Wave) cases.

partition interface are kept fixed while still allowing the true geometric boundaries to be modified. Multiple passes of the mesh adaptation algorithm are used to migrate the previous partition interfaces to the interior of the partition so as to ensure the global mesh is adapted before returning the mesh to the solver. The algorithm was verified to be (1) scalable and (2) stable, in which (1) good speedup was observed as the number of processors was increased and (2) the produced meshes are metric-conforming for 3d and 4d problems. In the future, we will investigate the performance of the algorithm with larger numbers (hundreds or thousands) of processors.

Future work involves integrating the parallel mesh adaptation algorithm with a  $3d + t$  numerical simulation tool so as to perform large-scale mesh adaptation for time-dependent partial differential equations in 3d [49, 50, 51, 52]. Furthermore, we hope to apply the algorithm to problems in which the domain geometry is moving in time, particularly if the domain topology is changing. A tool capable of generating an initial 4d mesh within such a domain would first be required.

The code associated with this work is publicly available in the LGPL-licensed software package: <https://gitlab.com/philipclaude/avro>

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1827373.

## References

- [1] Yano M., Modisette J.M., Darmofal D.L. “The Importance of Mesh Adaptation for Higher-Order Discretizations of Aerodynamic Flows.” *20th AIAA Computational Fluid Dynamics Conference*, 3852. Jun. 2011
- [2] Slotnick J., Khodadoust A., Alonso J., Darmofal D.L., Gropp W., Lurie E., Mavriplis D.J. “CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences.” Tech. Rep. NASA/CR-2014-218178, 2014
- [3] Bangerth W., Rannacher R. “Finite Element Approximation of the Acoustic Wave Equation: Error Control and Mesh Adaptation.” *East-West Journal of Numerical Mathematics*, vol. 7, no. 4, 263–282, 1999
- [4] Hartmann R. “Adaptive FE Methods for Conservation Equations.” H. Freistühler, G. Warnecke, editors, *Hyperbolic Problems: Theory, Numerics, Applications*, vol. 141 of *International Series of Numerical Mathematics*, pp. 495–503. Feb. 2001
- [5] Erickson J., Guoy D., Sullivan J., Üngör A. “Building Space-Time Meshes over Arbitrary Spatial Domains.” *Engineering with Computers*, vol. 20, 342–353, 08 2005
- [6] Behr M. “Simplex Space-Time Meshes in Finite Element Simulations.” *International Journal for Numerical Methods in Fluids*, vol. 57, no. 9, 1421–1434, 7 2008
- [7] Bangerth W., Geiger M., Rannacher R. “Adaptive Galerkin Finite Element Methods for the Wave Equation.” *Computational Methods in Applied Mathematics*, vol. 10, no. 1, 3–48, 2010
- [8] Fidkowski K.J. “Output-Based Space-Time Mesh Optimization for Unsteady Flows Using Continuous-in-Time Adjoints.” *Journal of Computational Physics*, vol. 341, no. 15, 258–277, 2017

- [9] Yano M. *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*. PhD thesis, Massachusetts Institute of Technology, Jun. 2012
- [10] Yano M., Darmofal D.L. “An Optimization-Based Framework for Anisotropic Simplex Mesh Adaptation.” *Journal of Computational Physics*, vol. 231, no. 22, 7626–7649, Sep. 2012
- [11] Jayasinghe S. *An Adaptive Space-Time Discontinuous Galerkin Method for Reservoir Flows*. PhD thesis, Massachusetts Institute of Technology, Jun. 2018
- [12] Jayasinghe S., Darmofal D.L., Burgess N.K., Galbraith M.C., Allmaras S.R. “A Space-Time Adaptive Method for Reservoir Flows: Formulation and One-Dimensional Application.” *Computational Geosciences*, vol. 22, no. 1, 107–123, Feb. 2018
- [13] Caplan P.C. *Four-dimensional Anisotropic Mesh Adaptation for Spacetime Numerical Simulations*. PhD thesis, Massachusetts Institute of Technology, Jun. 2019
- [14] Yano M., Darmofal D.L. “A Fully-Unstructured Space-time Adaptive Method for Wave Propagation.” *Computer Methods in Applied Mechanics and Engineering*, 2014
- [15] Chrisochoides N.P. “Telescopic Approach for Extreme-Scale Parallel Mesh Generation for CFD Applications.” *46th AIAA Fluid Dynamics Conference*, 3181. 2016
- [16] Tsolakis C., Chrisochoides N., Park M., Loseille A., Michal T. “Parallel Anisotropic Unstructured Grid Adaptation.” *2019 AIAA Science and Technology Forum*, 2019–1995. 2019
- [17] Michal T., Krakos J. “Anisotropic Mesh Adaptation through Edge Primitive Operations.” *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 159. Jan. 2012
- [18] Loseille A., Alauzet F., Menier V. “Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes.” *Computer-Aided Design*, vol. 85, 53 – 67, 2017
- [19] Tsolakis C., Drakopoulos F., Chrisochoides N. “Sequential Metric-Based Adaptive Mesh Generation.” *2018 Modeling, Simulation, and Visualization Student Capstone Conference*. Suffolk, VA, Apr. 2018
- [20] Park M.A., Darmofal D.L. “Parallel Anisotropic Tetrahedral Adaptation.” *46th AIAA Aerospace Sciences Meeting and Exhibit*, 917. 2008
- [21] Coupez T., Dignonnet H., Ducloux R. “Parallel Meshing and Remeshing.” *Applied Mathematical Modelling*, vol. 25, no. 2, 153–175, 2000
- [22] Coupez T. “Génération de Maillage et Adaptation de Maillage par Optimisation Locale.” *Revue Européenne des Éléments Finis*, vol. 9, no. 4, 403–423, 2000
- [23] Loseille A., Löhner R. “Cavity-Based Operators for Mesh Adaptation.” *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 152. 2013
- [24] Caplan P.C., Haines R., Darmofal D.L., Galbraith M.C. “Extension of Local Cavity Operators to  $3d + t$  Spacetime Mesh Adaptation.” *2019 AIAA Science and Technology Forum*. AIAA, 2019
- [25] Caplan P.C., Haines R., Darmofal D.L., Galbraith M.C. “Four-Dimensional Anisotropic Mesh Adaptation.” *Computer-Aided Design*, vol. 129, 102915, 2020
- [26] Haines R., Dannenhoffer J. “The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry.” *21st AIAA Computational Fluid Dynamics Conference*, 2013–3073. 2013
- [27] Haines R., Dannenhoffer J. “EGADSLite: A Lightweight Geometry Kernel for HPC.” *AIAA Aerospace Sciences Meeting*, 2018–1401. 1 2018
- [28] Ibanez D., Barral N., Krakos J., Loseille A., Michal T., Park M. “First Benchmark of the Unstructured Grid Adaptation Working Group.” *Procedia Engineering*, vol. 203, 154 – 166, 2017. 26th International Meshing Roundtable
- [29] Park M.A., Balan A., Anderson W.K., Galbraith M.C., Caplan P.C., Carson H.A., Michal T., Krakos J.A., Kamenetskiy D.S., Loseille A., Alauzet F., Frazza L., Barral N. “Verification of Unstructured Grid Adaptation Components.” , no. 2019–1723, 2019
- [30] Dignonnet H., Silva L., Coupez T. “Massively Parallel Computation on Anisotropic Meshes.” *Proceedings of the 6th International Conference on Adaptive Modeling and Simulation*, pp. 199–211. 2013
- [31] Dignonnet H., Coupez T., Laure P., Silva L. “Massively Parallel Anisotropic Mesh Adaptation.” *The International Journal of High Performance Computing Applications*, vol. 33, no. 1, 3–24, 2017

- [32] Digonnet H., Coupez T., Laure P., Silva L. “Massively Parallel Anisotropic Mesh Adaptation.” *The International Journal of High Performance Computing Applications*, vol. 33, no. 1, 3–24, 2019
- [33] Lachat C., Pellegrini F., Dobrzynski C. “PaMPA: Parallel Mesh Partitioning and Adaptation.” *21st International Conference on Domain Decomposition Methods (DD21)*. INRIA Rennes-Bretagne-Atlantique, Rennes, France, Jun. 2012
- [34] Lachat C. *Conception et Validation d’Algorithmes de Remaillage Parallèles à Mémoire Distribuée basés sur un Remaillieur Séquentiel*. PhD thesis, Université de Nice-Sophia Antipolis, Dec. 2013
- [35] Lachat C., Dobrzynski C., Pellegrini F. “Parallel Mesh Adaptation using Parallel Graph Partitioning.” *5th European Conference on Computational Mechanics*, vol. 3, pp. 2612–2623. CIMNE - International Center for Numerical Methods in Engineering, Jul. 2014
- [36] Lachat C., Pellegrini F., Dobrzynski C., Staffebach G. “Fast Parallel Remeshing for Accurate Large-Eddy Simulations on Very Large Meshes.” Research Report RR-9133, Inria Bordeaux Sud-Ouest, Dec. 2017
- [37] Dobrzynski C. “MMG3D: User Guide.” Technical Report RT-0422, INRIA, Mar. 2012
- [38] Loseille A., Menier V., Alauzet F. “Parallel Generation of Large-size Adapted Meshes.” *Procedia Engineering*, vol. 124, 57–69, 2015. 24th International Meshing Roundtable
- [39] Loseille A. “Unstructured Mesh Generation and Adaptation.” *Handbook of Numerical Methods for Hyperbolic Problems*, vol. 18 of *Handbook of Numerical Analysis*, chap. 10, pp. 263 – 302. 2017
- [40] Drakopoulos F., Tsolakis C., Chrisochoides N.P. “Fine-Grained Speculative Topological Transformation Scheme for Local Reconnection Methods.” *AIAA Journal*, vol. 57, no. 9, 4007–4018, 2019
- [41] Alauzet F., Li X., Seol E.S., Shephard M.S. “Parallel Anisotropic 3D Mesh Adaptation by Mesh Modification.” *Engineering with Computers*, vol. 21, no. 3, 247–258, May 2006
- [42] Schloegel K., Karypis G., Kumar V. “Parallel Static and Dynamic Multi-Constraint Graph Partitioning.” *Concurrency Computation*, vol. 14, no. 3, 219–240, Mar. 2002
- [43] Chevalier C., Pellegrini F. “PT-Scotch: A Tool for Efficient Parallel Graph Ordering.” *Parallel Computing*, vol. 34, no. 6, 318–331, 2008. Parallel Matrix Algorithms and Applications
- [44] Devine K.D., Boman E.G., Heaphy R.T., Hendrickson B.A., Teresco J.D., Faik J., Flaherty J.E., Gervasio L.G. “New Challenges in Dynamic Load Balancing.” *Applied Numerical Mathematics*, vol. 52, no. 2, 133–152, 2005. ADAPT ’03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation
- [45] Lusk E., Gropp W. “The MPI Message-Passing Interface Standard: Overview and Status.” J. Dongarra, G. Joubert, L. Grandinetti, J. Kowalik, editors, *High Performance Computing*, vol. 10 of *Advances in Parallel Computing*, pp. 265–269. North-Holland, 1995
- [46] Tsolakis C., Chrisochoides N., Park M.A., Loseille A., Michal T. “Parallel Anisotropic Unstructured Grid Adaptation.” *AIAA Journal*, vol. 0, no. 0, 1–13, 0
- [47] Carson H.A. *Provably Convergent Anisotropic Output-based Adaptation for Continuous Finite Element Discretizations*. PhD thesis, Massachusetts Institute of Technology, Jun. 2020
- [48] Carson H.A., Huang A.C., Galbraith M.C., Allmaras S.R., Darmofal D.L. “Anisotropic Mesh Adaptation for Continuous Finite Element Discretization through Mesh Optimization via Error Sampling and Synthesis.” *Journal of Computational Physics*, vol. 420, 109620, 2020
- [49] Galbraith M.C., Allmaras S.R., Darmofal D.L. “A Verification Driven Process for Rapid Development of CFD Software.” *53rd AIAA Aerospace Sciences Meeting*, 2015-0818. January 2015
- [50] Nishikawa H., Padway E. *An Adaptive Space-Time Edge-Based Solver for Two-Dimensional Unsteady Inviscid Flows*
- [51] Frontin C.V., Walters G.S., Witherden F.D., Lee C.W., Williams D.M., Darmofal D.L. “Foundations of Space-Time Finite Element Methods: Polytopes, Interpolation, and Integration.” *Applied Numerical Mathematics*, vol. 166, 92–113, 2021
- [52] Williams D.M., Frontin C.V., Miller E.A., Darmofal D.L. “A family of symmetric, optimized quadrature rule for pentatopes.” *Computers & Mathematics with Applications*, vol. 80, no. 5, 1405–1420, 2020



# SHRINK WRAP MESH GENERATION USING MORPHOLOGICAL OPERATORS WITH SELECTED APPLICATIONS

Vijai Kumar Suriyababu<sup>1,‡</sup>

Cornelis Vuik<sup>1</sup>

Matthias Möller<sup>1</sup>

<sup>1</sup>*Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands,  
{v.k.suriyababu,c.vuik,m.moller}@tudelft.nl*

<sup>‡</sup> *Corresponding Author*

## ABSTRACT

Triangulated meshes discretized from commercial CAD applications often possess a considerable level of complexity. However, when conducting external aerodynamics simulations at an earlier design stage, these meshes are way too complex and contain complex features and topological holes. We propose a practical and fast algorithm to shrink wrap triangulated surfaces with the sole intent of topology and surface simplification. Building upon the concepts of mathematical morphology and newer advancements in geometry processing, such as generalized winding numbers, we show that it is possible to build a straightforward and robust algorithm that can guarantee genus-zero surfaces. Our approach uses a Cartesian background mesh (fixed and adaptive) to approximate an input triangulated surface's interior and exterior volume. We use an octree data structure for adaptive mesh refinement. Although we demonstrate our algorithm exclusively on triangulated meshes, they are equally applicable to general polyhedral meshes. They are also well suited for handling point clouds (oriented and unoriented), and we show some examples of the same with some unoriented point clouds. We built our algorithms with a wide variety of applications in mind. However, we showcase the applicability of our algorithms for aerodynamic simulations, fluid volume extraction, and surface simplification. We also emphasize the practicality and ease of implementation of the proposed algorithms. We also compare our algorithms with existing literature.

**Keywords:** mesh simplification, shrink wrapping, boolean operations, fluid volume extraction

## NOMENCLATURE

$\mathcal{M}$	Triangulated mesh (or surface)
$\sigma$	Structuring element
$\sigma_r$	Structuring element radius
$T_l$	Topological sphere level
$\omega$	Solid angle for a query point with respect to an input mesh

## 1. INTRODUCTION

The shrink-wrapping algorithm can be a helpful tool for remeshing triangulated (or any polyhedral) surfaces. Most algorithms take a volumetric approach to

solve the problem. They work by projecting a voxelized approximation of the input surface onto itself. A few papers in the literature focus on this problem, and most work is accomplished in the industry. Attene et al. [1] gives a very detailed comparison of different mesh repair algorithms. Their review article's "Global repair section" gives an excellent overview of the state-of-the-art algorithms with a Global mesh repair and simplification approach. A key highlight of this discussion is that all authors take a volumetric approach to the problem. However, none of these explicitly solve the problem of shrink-wrapping with CAE simulations in mind. They are generic mesh simplification approaches, with the primary focus being

computer graphics applications. Some notable contributions are from Esteve et al.[2] and Nooruddin et al.[3], who also take volumetric approaches. Esteve et al.[2] shrink a discrete membrane to re-mesh surface meshes and point clouds, whereas Nooruddin et al.[3] take a strict morphological approach to the problem. One major drawback in their works is a lack of control over the surface genus. In the case of Nooruddin et al.[2], they oversimplify the meshes in all of the examples instead of our shrink-wrap mesh simplification. They also do not guarantee a manifold mesh as output, thus rendering their results unsuitable for numerical simulation. Y. K. Lee et al. [4] gave a good summary of the existing techniques that are tailored to the problem of shrink-wrapping with engineering applications in mind. They highlight the effectiveness of these algorithms in closing gaps and removing interior parts of complex geometry along with their potential as remeshing algorithms. They also show that gap detection and bridging are usually achieved with the help of poor / coarser voxelization. There is often a requirement to intersect these voxels with the input surface mesh, increasing computational costs. Some algorithms [5] require explicit tolerance values for the gaps and holes, which could be advantageous in some applications. However, we explicitly focus on fully automated genus simplification. Hence, we propose an algorithm that can inherently close all the gaps in triangulated surfaces and remove all internal structures. The significant contribution in our work is an efficient way of computing genus simplified offset surface with the help of morphological operators. Our algorithms will guarantee an outcome even in the case of imperfect geometries. Imperfections can range from missing triangles to non-manifold edges or completely separated components. We demonstrate the same in our numerical experiments. We also extend the existing shrink-wrap algorithm for selective genus control. An existing semi-heuristic algorithm that we developed [6] for hole detection is used to control the closing operations so that only selective holes are closed.

## 2. MORPHOLOGICAL OPERATORS

Mathematical morphology is a vibrant subject that finds typical applications in the area of image processing [7]. However, it has been extended to many other application areas, including three-dimensional geometry processing [8]. Broadly, the subject of mathematical morphology is composed of four operators.

1. Erosion ( $\mathcal{M} \ominus \sigma$ )
2. Dilation ( $\mathcal{M} \oplus \sigma$ )
3. Opening ( $(\mathcal{M} \ominus \sigma) \oplus \sigma$ )
4. Closing ( $(\mathcal{M} \oplus \sigma) \ominus \sigma$ )

Given a triangulated mesh  $\mathcal{M}$ , a structuring element  $\sigma$  with a radius of  $\sigma_r$ , the erosion operator erodes a surface iteratively for any given number of steps. The dilation operator does the exact opposite and adds to the given mesh. Our work is similar to the approach of Zhen Chen et al. [9] in that they use morphological operations to compute discrete surface offsets. Since our focus here is a restricted form of mesh and topology simplification, we take a lot of freedom to interpret these morphological operators. We also do not compute exact offset surfaces. Instead, we calculate a simplified/approximated offset surface that is only useful for topological simplification. We also use Octrees to represent all operators, allowing the mesh to be reused for numerical simulation. Finally, we introduce a topological sphere level parameter similar to the scaled structuring element radius given by  $\sigma_r$  (scaled by the offset distance). We encourage reading upon the work of Silvia Sellán et al. [10] for a more direct interpretation of these operators in the context of three-dimensional geometry processing. They explore a surface-only approach that only modifies selective areas of a surface mesh. However, as they point out, their work suffers from the flaws of surface flows and isn't suitable for shrink wrapping applications. In our work, we open a surface iteratively until the surface has a spherical topology and then close it by iterative erosion. We explain the different morphological operators in the context of mesh generation in detail in the subsequent sections.

### 2.1 Mathematical morphology of surface meshes

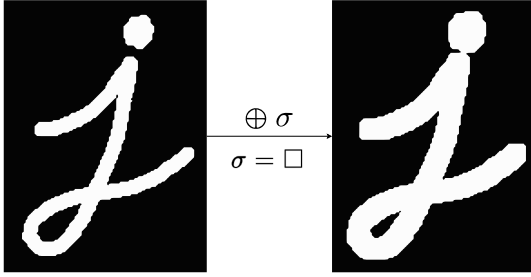
Morphological operators in our present investigation do not differ very much from their image processing counterparts. In image processing, one represents morphological operators on simple two-dimensional grids. For example, a binary image can be represented on a Cartesian mesh with binary mask values (0 and 1). Therefore, the morphological operators in image processing can be considered as two-dimensional simplification of our morphological operators. First, we represent a surface mesh in a fixed or adaptively refined Cartesian mesh with binary values to identify the geometry's boundary. Then, we perform the morphological closing operation of this approximated geometry followed by surface extraction and projection. One key difference with our erosion operator is that we do not erode the geometry beyond its original boundary to preserve the internal volume. This is clearly shown in subsequent sections.

Let us consider the dilation and erosion operators on binary images and their geometric counterparts. Since other operators such as opening and closing can be built as a combination of these two, it is enough to

understand these two operators.

## 2.2 Dilation

We first dilate the given image with a square as a structural element. This is the most natural choice since the images are represented as pixels. Furthermore, this approach would only require adjacencies at the edge level to find neighbours from a computational viewpoint. Once the boundary of the image is identified, one step of dilation becomes a straightforward problem of finding the neighbouring faces of the border. The key detail in this approach is that the neighbours should be chosen in the positive normal direction of the boundary. Until a required criterion is achieved, this process can then be repeated for any number of steps.



**Figure 1:** Dilation operation on a binary image

This operation remains the same in mesh processing. As stated earlier, we mark the boundary cells in a Cartesian mesh. Then the dilation operation for mesh processing is a natural extension of image processing to a three-dimensional problem. This operation is explained in detail with algorithmic workflows for computing a dilated surface from a triangulated mesh.

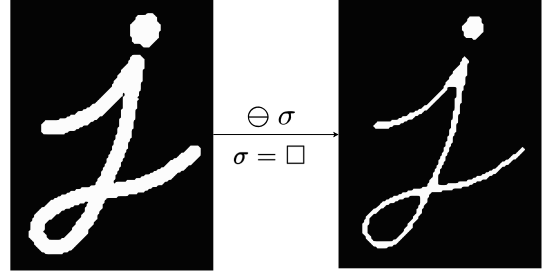
### 2.2.1 Erosion

The erosion operator also starts from the boundary of a given image. First, however, it finds the neighbours of the border in the negative normal direction. So these cells would usually be part of the image itself. A pivotal contrast to the dilation approach is that an image cannot be eroded infinitely. Since at some point, the erosion operation reaches a singularity.

In the case of shrink wrapping, erosion operation should not destroy the internal volume of a surface mesh. Hence, an erosion operation usually stops at the boundary of a surface mesh.

### 2.2.2 Closing

The closing operator combines the dilation and erosion operation. In the field of computer vision and image

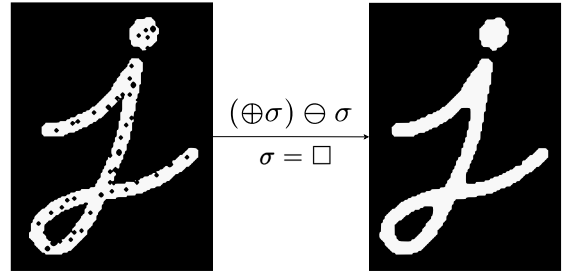


**Figure 2:** Erosion operation on a binary image

processing, this serves as a tool to close holes (or missing pixels) in a binary image. We perform the same operation on three-dimensional data for closing holes in our algorithms. To provide an easy comparison towards mesh processing, we have extracted a three-dimensional surface of the same geometry (by extruding the contour in the Z direction) and performed the closing operation on the same. The results for the same can be seen in figure 4.

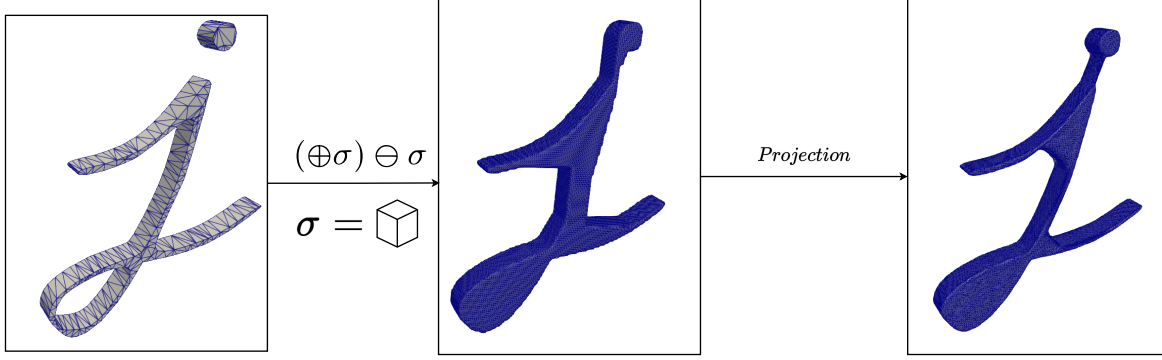
The two differences from the image processing approach are as follows

- We do not erode beyond the boundary of the input geometry
- We project the eroded geometry onto the input geometry



**Figure 3:** Closing operation on a binary image (Black blobs on the left indicate missing pixels). They are equivalent to holes or missing triangles in a surface mesh.

It can be observed in figure 4 that a closing operation on a surface not only closes the holes caused by missing triangles. It also seals the topological holes in a mesh. We leverage this benefit for shrink wrapping surface meshes for external aerodynamic simulations. However, the straightforward closing operation on its own is not suitable since it has no stopping criterion. Therefore, our workflow introduces a series of algorithms and data structures such as octrees to use these morphological operators efficiently for shrink



**Figure 4:** Closing operation on a three dimensional surface. This geometry is equivalent to the two dimensional binary images in figure 3. Closing operation is performed on the left most geometry and then projected onto the ground truth.

wrapping surface meshes. We explain these in detail in the subsequent section.

### 3. BASIC WORKFLOW

Our algorithm has the following basic components

1. Conversion of Boundary representation(B-Rep) to a volumetric representation (V-Rep)
2. Computation of signed distance function
3. Dilation of the input surface for a given topological sphere level
4. Erode the dilated surface to obtain a topologically hole-free offset surface
5. Iteratively project and smooth the dilated surface
6. Remesh to improve triangle quality (Optional)

We explain these individual components in detail along with the respective algorithms in the subsequent sections.

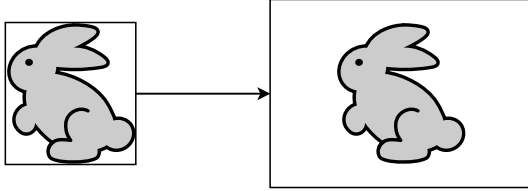
#### 3.1 B-Rep to V-Rep

It was shown earlier that geometry information needs to be encoded as binary images for efficient computation of morphological operators in computer vision applications. This translates to a boolean value in every voxel of a Cartesian mesh<sup>1</sup> in three dimensions. However, it would require intersecting the surface mesh with the Cartesian mesh. This would only work for

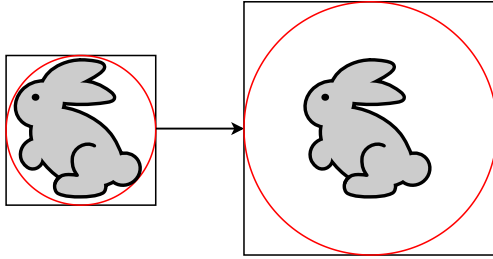
<sup>1</sup>We use the terms Cartesian mesh and octree interchangeably throughout the paper. The reader should be aware that all the computations are performed on an octree, which we consider a special kind of Cartesian mesh.

watertight surface meshes and lead to some inaccuracies in the case of degenerate geometries. Hence, we do not physically intersect the input geometry with the Cartesian mesh. Since we only need a scalar field to distinguish the inside and outside of the geometry. The usual workflow for Cartesian mesh generation starts with a bounding box computation as shown in figure 5. These can be an axis-aligned or oriented-bounding box. In either case, the generated Cartesian mesh would not be very beneficial for morphological operations. Morphological operators such as erosion and dilation are applied in successive layers. For example, the dilation operator starts from the boundary of a surface and dilates the surface one layer after another, as shown in figure 1. Since we use a cube or a voxel as a structuring element, it is easier to perform these operations successively if the geometry sits approximately in the centre of the Cartesian mesh. If the geometry is moved to the origin of the octree, a dilation operation may not completely dilate the entire surface in a given step. First, the mini ball algorithm [11] is used to obtain a tightly fitting sphere of an input geometry as shown in figure 6. Then we compute a bounding box for this sphere with a specified offset threshold to ensure that our geometry always sits precisely at the centre of our voxelization. Next, we refine all the cells inside the tightly fitting sphere as shown in figure 7. Post refinement, the generalized winding number approach [12] helps distinguish the cells inside and outside the geometry. The generalized winding number algorithm gives a solid angle value at every vertex in the octree mesh. This value is thresholded to mark the cells in the octree as inside, outside or boundary cells. This refinement allows us to get a more accurate surface description during the segmentation process. Spherical refinement also limits the inside-outside queries to the cells within the sphere. As a result, we do not need to query the generalized winding number for cells outside the sphere, thereby

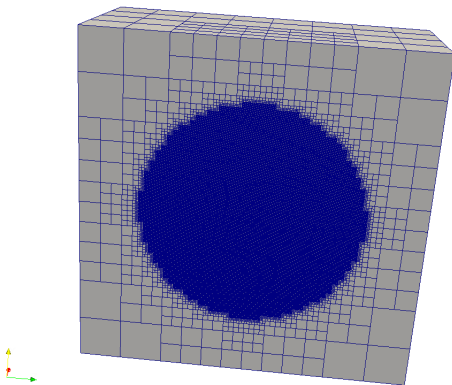
saving computational time. Finally, we also ensure a 2:1 refinement in our octree for all elements in our workflow. The mesh generation approach referenced in algorithm 1 can be used for any kind of numerical simulation irrespective of the rest of the workflow.



**Figure 5:** Normal bounding box computation (In case of many geometries, a manual offset threshold may be required for ensuring there are enough layers of mesh for morphological operations. The threshold might also be different for different directions.)



**Figure 6:** Spherical bounding box computation (In most cases, a threshold of 2 or 3 times the size of the bounding sphere is enough for all morphological operations. The scaling will be uniform irrespective of the geometry since only the sphere is scaled and the bounding box is always a perfect cuboid.)



**Figure 7:** Spherical Refinement

---

#### Algorithm 1: B-Rep to V-Rep

---

**Result:** Voxelized mesh where every cell has a scalar associated with it (inside / outside)

Initialise Surface;

Compute a tight bounding sphere using the mini ball algorithm and store its radius and centre ;

Calculate the bounding box of the sphere, which is offset at a user-specified distance (*2.0 in our experiments*);

Initialize a Cartesian mesh with a specified cell size or number of cells (*64 \* 64 \* 64 in all of our experiments*);

**forall** Cells of Cartesian mesh **do**

**if** Cell inside bounding sphere **then**

        Mark for refinement;

**end**

**else**

        Mark for coarsening;

**end**

**end**

**forall** Cells in bounding sphere **do**

    Compute the Generalized winding number

    (This indirectly gives us the solid angle for all the cells in the octree);

**end**

Mark cells outside bounding sphere as outside and store this in the respective cells;

**forall** Cells in bounding sphere **do**

**if** Solid angle is higher than 0.9 steradians (based on our experimental observation)

**then**

            Mark the cell as inside and store this in the respective cell;

**end**

**else**

        Mark the cell as outside and store this in the respective cell;

**end**

**end**

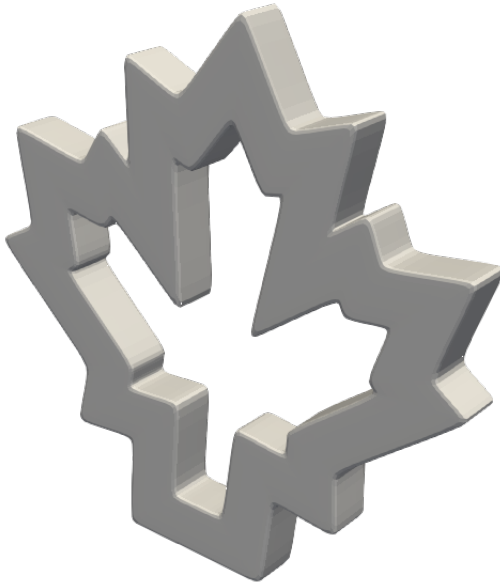
---

### 3.2 Computation of signed distance function

It is evident that once the cells of the octree are classified into inside and outside (using any approach such as generalized winding numbers in our case), an artificial signed distance function can be bestowed upon the voxelization as shown in figure 9. We rely on Generalized winding numbers since they are swift even on a CPU only computational environment and are immune to imperfections in the input surface to a large degree. A brief overview of this approach can be seen in appendix A. However, for the rest of the algorithmic workflow, one only needs to categorize the cells in the octree as inside or outside cells. These will

be used to build an approximate surface boundary which can be used for morphological operations described in the consequent sections. Our experimental observation has shown that a solid angle value of 0.9 steradians indicates cells inside a surface mesh, and everything else can be marked as outside. The bounding sphere computed in the mesh generation algorithm can be used to automatically mark all the cells outside the sphere as outside cells.

We use the term artificial since we do not compute the exact distance here. We only use an integer that indicates a particular voxel's relative position with its respective boundary voxel. As will be evident later, we do not need an exact signed distance field for computing a Genus simplified offset surface. A similar approach has been used by other researchers [13] to calculate intersection-free offset surfaces. We achieve the same by outward propagation from the zero level set voxels. This outward propagation is done along the normal outward direction of the surface mesh. Since we mark all the cells in the octree as inside or outside, zero level set voxels or boundary voxels can be determined by finding cells that contain faces that are part of both inside and outside cells. As opposed to the usual approaches, which intersect the surface mesh with the octree mesh, our proposed method is highly computationally efficient. All the morphological operations are explained with the help of a maple leaf geometry shown in figure 8.



**Figure 8:** Maple leaf geometry



**Figure 9:** Artificial signed distance function of a maple leaf (Computed using our approach)

### 3.3 Dilation driven approximated offsets

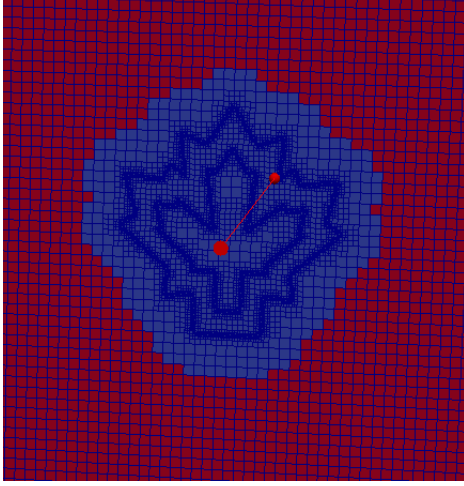
In the previous section, we proposed a straightforward method to determine the zero level set or boundary voxels of a given surface mesh inside an octree. We already established that a dilation operation followed by an erosion operation performed in a sequence leads to the morphological closing operation as shown in figure 3. Our investigation also reveals that one does not need to dilate the boundary voxels across the entire voxelization. Instead, we only need to dilate the input surface until we achieve a spherical topology. Here, we use a user-specified parameter called “**Topological sphere level**”. This parameter is the only user-controlled input in the algorithm, and the choice of topological sphere level dictates the number of outward propagation levels as shown in algorithm 2. The bigger the hole in the geometry, the larger the topological sphere level. Effects of different Topological sphere levels are clearly shown in the numerical experiments. For example, the dilated maple leaf geometry can be seen in figure 10. It is clearly evident that a spherical topology is achieved after approximately 15 levels.

If complete automation is required from input to projection, the topological sphere level can be ignored, and the geometry can be dilated to the maximum level.

### 3.4 Erosion of dilated offset surface

The dilated surface should now have a spherical topology, and it needs to be eroded towards the input surface. This process is similar to the dilation except for





**Figure 10:** Dilated Maple leaf geometry with a spherical topology

---

**Algorithm 2:** Dilation of the input surface

---

**Result:** Dilated surface stored in the voxelized mesh

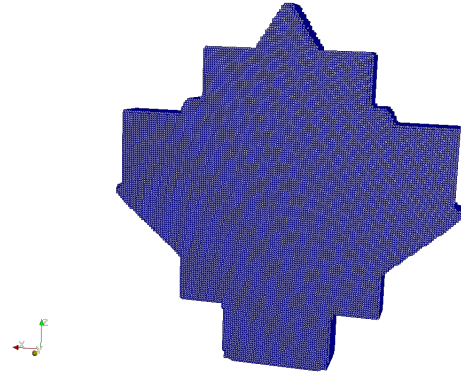
Initialize **interior\_cells** as **seed\_cells**;  
Initialize **current\_topological\_sphere\_level** to 0;  
**while** **current\_level**  $\leq$  **topological\_sphere\_level**  
  **do**  
    Initialize a **newer\_seeds\_cells\_id** vector;  
    **forall** **cells** in **seed\_cells** **do**  
      **forall** **cell\_neighbours** in **voxelized\_mesh** **do**  
        **if** **Neighbour** is outside **cell** **then**  
          Add neighbour to  
          **newer\_seeds\_cells\_id**;  
        **end**  
      **end**  
    Set **newer\_seeds\_cells\_id** as **seed\_cells**;  
    Increment **current\_topological\_sphere\_level**;  
    **if** **current\_topological\_sphere\_level** eq  
      **topological\_sphere\_level** **then**  
      Store these cells as  
      **topological\_sphere\_cells**;  
    **end**  
  **end**  
**end**

---

the marching direction. The number of levels would be the same as the topological sphere level chosen during the previous step of the algorithm. Once eroded, this will give a hole-free approximation of the input geometry. Once the surface is eroded to the given “Topological Sphere Level”, the operation becomes straightforward. It is explained in detail in algorithm 3. The scalar field can be directly eroded until it hits the boundary voxels. This is where the erosion operation

differs from the erosion operation in computer vision algorithms. In the case of surface mesh, the geometry is never eroded beyond the boundary voxels for volume preservation. This approach is relatively simple since a manifold mesh can be easily extracted without the need for any additional algorithms [14].

The offset surface is still embedded inside a volumetric mesh as shown in figure 11, and a surface needs to be extracted. Due to the artificial nature of the signed levels in the volumetric mesh, it is easy to distinguish the region where the genus simplified offset meets the external offset surface. Surface extraction becomes a simple task with this information. This is similar to the approach proposed for identifying boundary voxels from inside and outside voxels. The detailed algorithm for surface extraction is listed in algorithm 4.



**Figure 11:** Eroded offset surface (Unsmoothed & Hole Free)

---

**Algorithm 3:** Erosion of the dilated offset surface

---

**Result:** Eroded surface stored in the voxelized mesh

Initialize **topological\_sphere\_cells** as **seed\_cell\_ids**;  
**forall** **topological\_sphere\_levels** **do**  
  Initialize a **newer\_seeds\_cells\_id** vector;  
  **forall** **cells** in **seed\_cells** **do**  
    **forall** **cell\_neighbours** in **voxelized\_mesh** **do**  
      **if** **Neighbour** is from lower  
      **topological\_sphere\_level** **then**  
      Add neighbour to  
      **newer\_seeds\_cells\_id**;  
    **end**  
  **end**  
  Set **newer\_seeds\_cells\_id** as **seed\_cells**;  
**end**  
**end**

Final **seed\_cells** form the basis for the Genus simplified offset surface;

---

---

**Algorithm 4: Surface Extraction**

---

**Result:** Genus simplified watertight surface  
Initialize `topological_sphere_cells` as `seed_cell_ids`;  
**forall** `topological_sphere_levels` **do**  
    Initialize a `newer_seeds_cells_id` vector;  
    **forall** `cells` in `seed_cells` **do**  
        **forall** `cell_neighbours` in `voxelized_mesh` **do**  
            **if** *Neighbour is from lower*  
                *topological\_sphere\_level* **then**  
                    Add neighbour to  
                        `newer_seeds_cells_id`;  
                **end**  
            **end**  
        Set `newer_seeds_cells_id` as `seed_cells`;  
    **end**  
**end**  
Final `seed_cells` form the basis for the Genus  
simplified surface;

---

### 3.5 Projection and smoothing

The eroded surface needs to be projected onto the input geometry. With practicality in mind, we chose a point cloud- based approach over direct projection on the triangulated surface. In realistic industrial geometries, the construction of an AABB tree is costly and time-consuming and leads to failure in many cases. Since we chose to allow input meshes that are not perfectly two-manifold, the point cloud-based approach will support a broader range of input meshes, including those that are entirely degenerate, as shown in the later section. We approximate the input geometry as a uniformly sampled point cloud and then construct a kD tree [15] on it. We also extended our algorithm for point cloud due to this projection approach. However, one can choose a more sophisticated method that projects directly onto the triangles in the input surface. This might produce erroneous results if the surface mesh is completely degenerate. We experimented with both a direct projection approach and the one using point cloud sampling as shown in algorithm 12 and the results were satisfactory for our point cloud approach.

We can find the nearest neighbour in this point cloud for every vertex in the eroded surface and move the vertex to this position. Unfortunately, results do not look good at this stage, and the mesh seems slightly tangled. However, our experiments show that a few cycles of Laplacian smoothing followed by projection will immediately provide better quality results, as seen in figure 5.

Our investigation also shows that the geometry can be double wrapped to achieve better quality results. In double wrapping, the final result from the first run

---

**Algorithm 5: Projection and Smoothing**

---

**Result:** Projected and smoothed mesh  
Sample a uniform point cloud on the input  
surface;  
Build a kD tree on the uniformly sampled point  
cloud;  
**forall** `vertices` in the *Extracted surface* **do**  
    Find the nearest vertex in the kD tree and  
    move the vertex;  
**end**  
**forall** `vertices` in the *projected mesh* **do**  
    Find one ring neighbourhood;  
    Average the position of the current vertex  
    with the vertices from the one ring  
    neighbourhood ;  
**end**

---

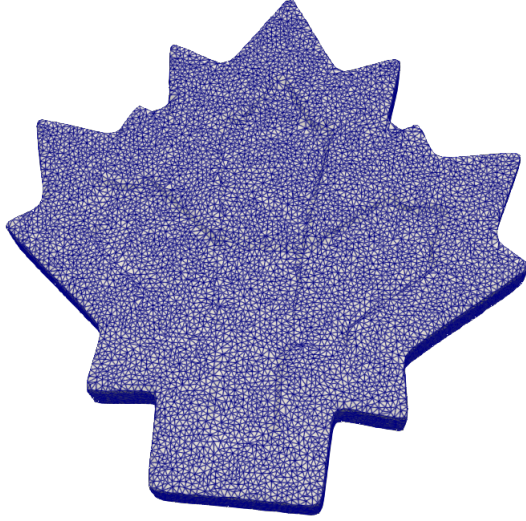


**Figure 12: Projected & Smoothed Mesh**

of the algorithm can be passed back onto the same workflow to produce a better quality approximation. The mesh at this stage is already analysis suitable. If required, an optional remeshing step can be included for improving the mesh quality further. The geometry practitioner is not required to follow our heuristic-based approach. They can choose any remeshing algorithm (commercial or public domain). However, the proposed algorithms provide satisfactory results in our investigation.

### 3.6 Remeshing and quality improvement (Optional)

This step is entirely optional. The projected surfaces are well suited for analysis, and we show the same in numerical experiments for various surface and volumetric PDEs. However, the smoothed surface may still have a few tangled edges and triangles with lousy quality. Hence, we propose a heuristic-based remesh-



**Figure 13:** Remeshed and quality improved maple leaf geometry

ing approach to improve triangle quality quickly. An overview of the same can be seen in algorithm 6. In all our numerical experiments, this approach seems to improve the mesh quality vastly.

---

**Algorithm 6:** Remeshing and quality improvement algorithm (Heuristic driven)

---

**Result:** Quality improved mesh

Compute the bounding box of the wrapped mesh and its diagonal length and offset the diagonal length by 0.005 (Based on our experimental observation);

Remove degenerated triangles;

Split long edges (edges longer than the diagonal length with our offset);

Store the **num\_vertices** at this level;

**while** **current\_num\_vertices**  $\neq$

**previous\_num\_vertices** **do**

    Collapse short edges (threshold of  $1e-06$ );

    Remove obtuse triangles (Above 150.0);

**end**

Resolve self intersections;

Remove duplicated faces;

Remove isolated vertices;

---

An additional stopping criterion can ensure the termination of the algorithm. One can also replace this heuristic with a more sophisticated remeshing algorithm such as the one driven by different error metrics. However, we understand that this heuristic-based approach is not very elegant. One can replace our algorithm with a black-box remeshing algorithm from libraries like CGAL[16]. Since we produce a genus

zero surface in most cases, spherical parameterization based remeshing approaches can also be considered an alternative. However, we found meshes at the projection stage suitable for numerical simulations. It is not within the scope of our work to investigate a dedicated remeshing approach. In fact, for the numerical experiments shown in the subsequent section, we use the geometries from the projection stage and ignore the remeshing routine altogether.

## 4. NUMERICAL EXPERIMENTS

We perform experiments on a wide variety of input geometries that help underscore the robustness of our algorithm. We noticed that even in the case of entirely ill-formed artefacts from industry, we could guarantee some form of a Genus simplified geometry. A wide variety of surfaces and their shrink-wrapped counterparts are shown in appendix B. In all cases, our algorithm produced a valid two-manifold surface mesh without any holes, and the Hausdorff distance was within a reasonable range (99% of the vertices are close to the original mesh).

### 4.1 Effect of topological sphere level

As stated earlier, “Topological sphere level” is the only parameter in the algorithm. In simpler terms, this is the number of layers the algorithm needs to travel along the positive normal direction of an input surface. It can be increased or decreased depending on the size of the biggest hole in the geometry. Since the algorithm is fast, the topological sphere level value can be chosen even on a trial and error basis. The algorithm could be allowed to propagate to the maximum possible level. However, this might significantly increase the algorithm’s run time, which is entirely unnecessary in our case. We show some examples of the same in section 5.2.1 and some of its pleasant side effects.

### 4.2 Effect on bad quality geometries

We show a car geometry in figure 16 which is missing most of its bottom. We use a coarser grid to produce a simplified approximation of the car geometry. It can be seen that our algorithm produces a tight wrap even in this case and simplifies the geometry. Since the offset computation does not require an exact segmentation of the geometry boundary, the hole free offset computation works even in such extremely poor quality geometries. The topological sphere level can be tuned on a trial and error basis for such geometries until the complete geometry is wrapped. In the case of skull geometry shown in figure 14, it has many non-manifold edges and has many disconnected components. There is no pre-processing requirement

on either of the geometries, and their shrink-wrapped results are shown in figure 15 a perfect two-manifold mesh without any leaks.

### 4.3 Boosting projection quality using external sources

We mentioned earlier that we double wrap the geometries to achieve a better projection quality. This is primarily due to the geometric structure of morphological erosion. It leads to a competitive projection which can be beneficial in many cases. For example, if the bottom is entirely missing, rather than failing to close the hole in the bottom, vertices are projected to the next closest area, which would be the boundary of the bottom hole. This ensures a good priori for the next wrapping stage. Hence, the double wrapping stage would provide a better distribution of triangles. This is an undesirable yet pleasant side effect of the competitive nature of the projection of the algorithm to stick to whatever comes first. However, if the geometry practitioner/end-user in the industry would like to have better conditioning for such holes / even topological holes, using external sources would help. In our earlier work [6], we proposed a semi heuristic algorithm to detect topological and geometric cavities in a triangulated mesh. Just like our present algorithm, it does not need a perfect two manifold mesh. We show later in section 5.2 that this can also be useful for selective Genus closing.

### 4.4 Runtime analysis

The algorithm proposed in this paper is memory bound, and memory usage depends entirely on the size of the octree and the number of triangles in the input mesh. Even though we mentioned the Topological sphere level as the only parameter in the algorithm, different geometric parameters related to octree could also be tuned towards requirements. All the investigations in the paper are carried out with a fixed initial grid size of 64 x 64 x 64 and 3 levels of spherical refinement, and it gives satisfactory results. The code was implemented in C++ with OpenMP parallelization, and the software is available as a binary (for Linux) for reproduction. However, the source code is not available due to commercial interests. All the results were obtained on a laptop with a 12 core Intel Core i7 CPU with 64 gigabytes of RAM. The program runtime (in minutes) versus the number of triangles in the input mesh is shown in figure 17. Our algorithm can scale linearly with the increasing number of triangles. The largest triangulation in our investigation had 23 million triangles, and the algorithm converged to a manifold surface in under 10 minutes.

## 5. APPLICATIONS AND VARIANTS

### 5.1 Surface PDE

Surface-based finite element methods[17] and Boundary element methods (BEM) have gained a lot of momentum in recent years among the geometry processing and engineering research community in general. Our goal here is to show the shrink-wrapped geometries suitability for surface PDE computations on standard benchmark problems. For the first example, we obtain selective eigenmodes of the Laplace equation (using Laplace Beltrami operator) based on the work of Vallet, B et al. [18]. For the second example, we solve for Geodesic distance using the heat method on our shrink-wrapped geometries. There are numerous PDE's and applications besides the ones shown in our investigation. We make no effort to suggest that we modified or improved the methodology in either of these examples. We only use these as an example to show that our geometries are suitable for analysis.

#### 5.1.1 Eigenmode decomposition of the Laplace equation

Eigenmode decomposition of the Laplace equation[18] can be helpful to represent functions on surfaces. It is most useful in real-time deformation problems. We use the famous Laplace-Beltrami operator to obtain its eigen decomposition on shrink-wrapped geometries. A generalized eigenvalue problem is of the form.

$$Ax = \lambda Bx \quad (1)$$

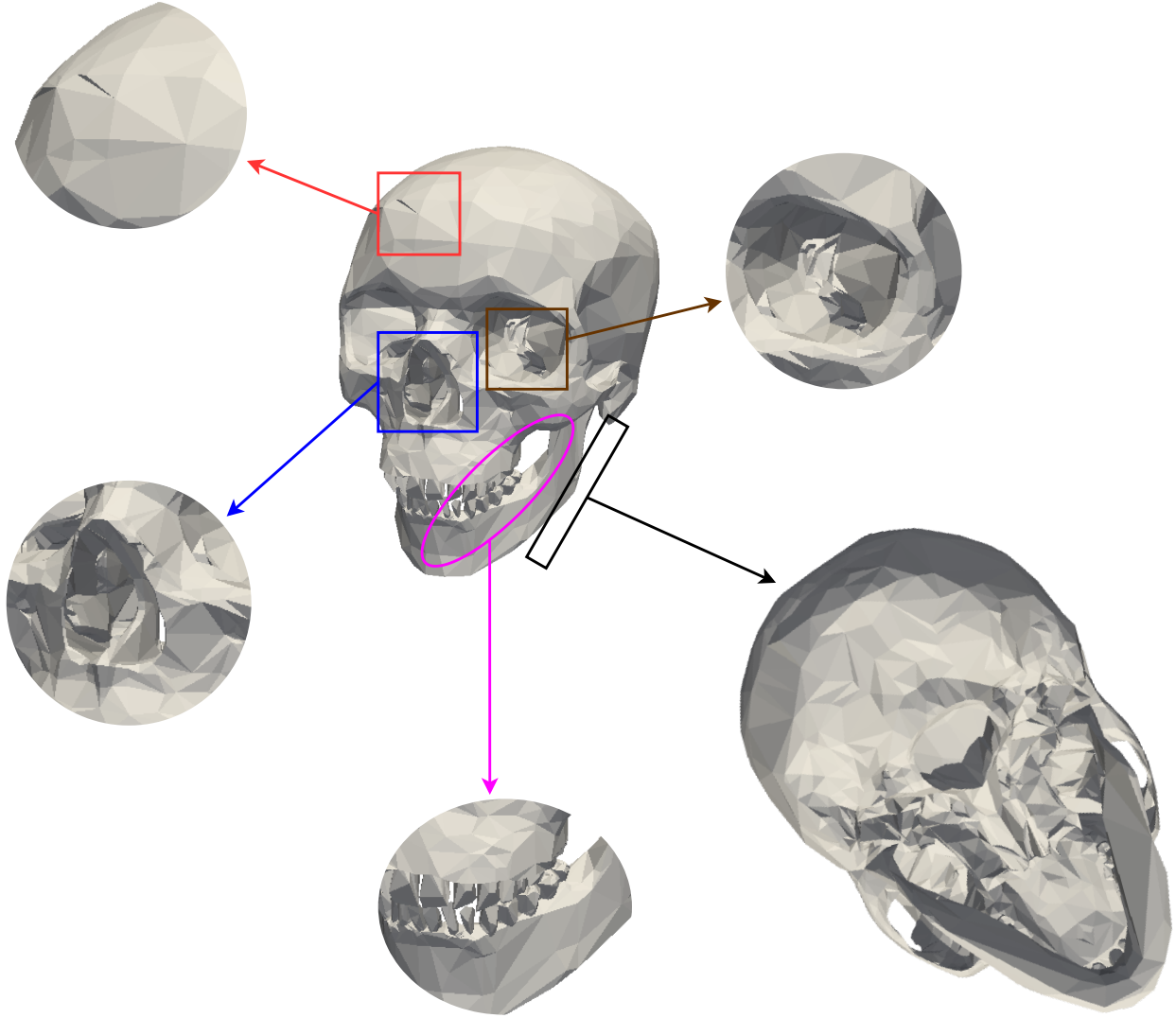
Here we use Laplace-Beltrami operator  $L$  for  $A$  and use per vertex mass matrix  $M$  of the shrink-wrapped surface as  $B$ . The Eigen mode functions can then be represented as follows[18]:

$$\mathbf{f} = \sum_{i=1}^n a_i \phi_i \quad (2)$$

where  $a_i$  represents the scalar coefficients and  $\phi_i$  represents the Eigen functions which satisfy  $\Delta \phi_i = \lambda_i \phi_i$ . Hence, we end up with an equation as follows

$$L\phi_i = \lambda_i M\phi_i \quad (3)$$

For low-frequency modes, it produces smooth and slowly changing functions on the shrink-wrapped mesh. For a more thorough understanding of the theory of manifold harmonics, it is best to refer to Vallet, B et al. [18]. We only chose Eigen decomposition as an example problem for showcasing the analysis suitability of our shrink-wrapped surfaces, and a detailed explanation of the underlying theory is beyond the scope



**Figure 14:** A human skull geometry with at least 15 non-manifold edges and many disconnected components. Various defects in a skull geometry (gaps, disconnected tooth, non-manifold edges and holes) are highlighted. It also includes the bottom view of the skull which is entirely deformed.

of our work. We solve for the first three Eigenmodes on a shrink-wrapped surface.

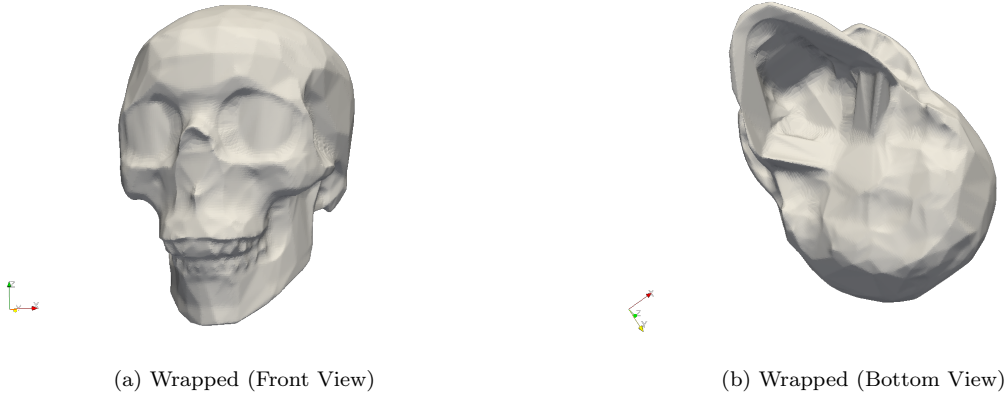
### 5.1.2 Geodesic distance (Heat method)

Geodesic distance has numerous applications in geometry processing and physics-driven simulations. We calculate the geodesic distance using the heat method[19] on our shrink-wrapped meshes. Geodesic distance contours at different point sources show a smooth transition of the heat contours across the sur-

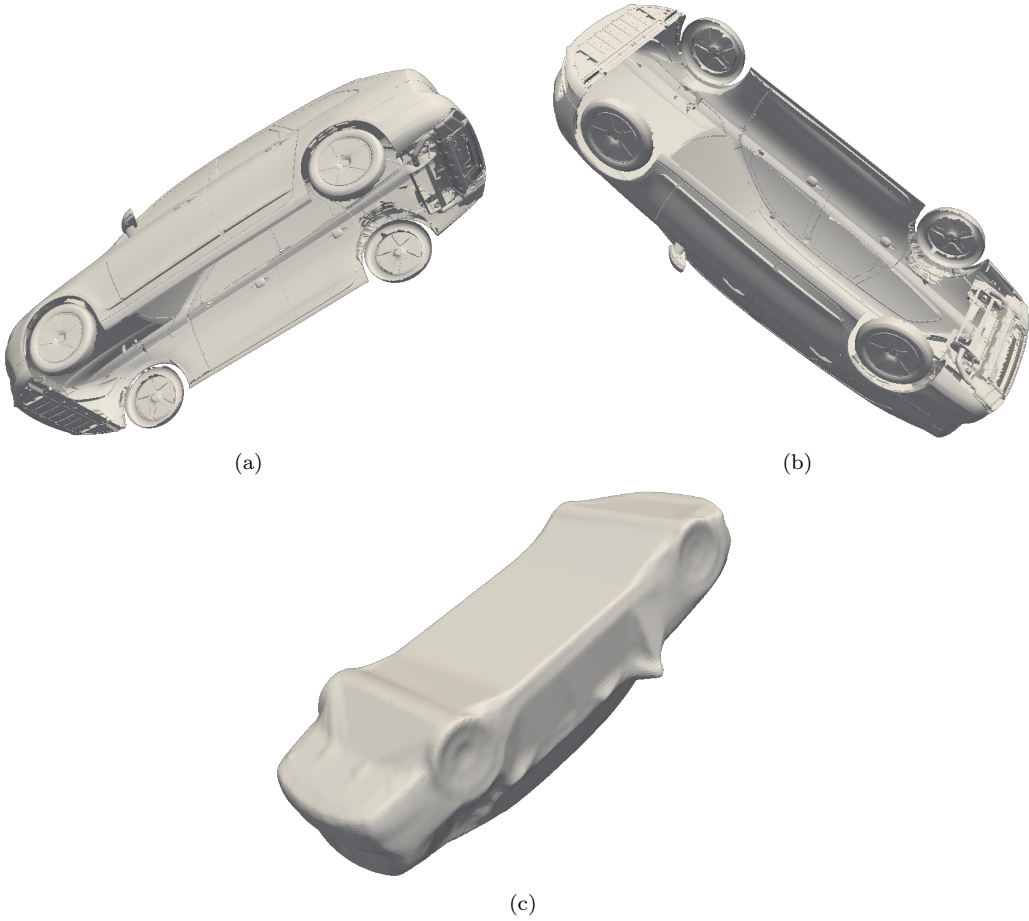
face. Furthermore, there is no noticeable noise anywhere in the scalar field, proving the algorithm's robustness in producing simulation capable meshes.

## 5.2 Selective Genus Closing

Almost all of the shrink wrapping algorithms are either used as remeshing (i.e. preserve the genus of the input mesh) or surface simplification algorithms (i.e. turn the input mesh into a topological sphere or Genus zero). However, there are scenarios where an indus-



**Figure 15:** Wrapped skull geometry (Watertight geometry with a genus zero)



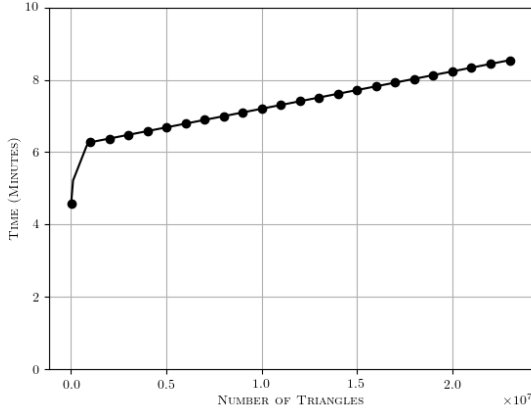
**Figure 16:** Bad quality geometries shrink wrapped (Car with hole) - Input mesh along with wrapped output mesh. It can be observed that the bottom of the car is completely closed.

trial practitioner is only interested in closing selective holes. We could not find any other works that directly address this problem. We propose two ways to do this

in our paper.

- Implicit Genus control





**Figure 17:** Run time comparison for increasing number of triangles

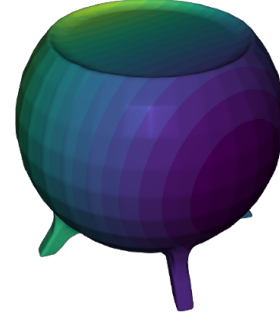
- Explicit Genus control

### 5.2.1 Implicit Genus control

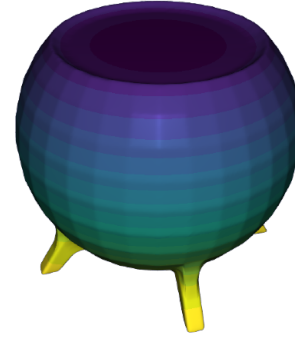
In this case, no additional algorithms are required. However, to achieve the necessary genus, it can be an iterative process. We leverage the topological sphere level's ability to control the genus implicitly. A lower value for the topological sphere level usually leads to incomplete closing of the geometry. This can be a desirable side effect in the case of selective Genus control. We have an example geometry in figure 20 below with a huge topological hole in the top and a smaller one in the bottom. The effects of different topological sphere levels are shown in the figure 21. It can be observed that a value of 50 yields a Genus zero surface; however, at a level 10, only the smallest hole in the mesh is closed.

### 5.2.2 Explicit Genus control

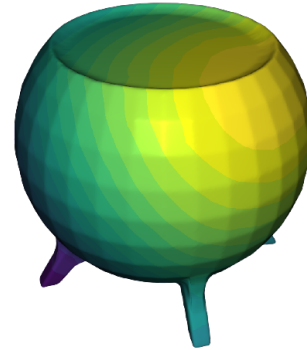
Explicit Genus control requires prior information about the topological holes in the mesh. Therefore, we use our topological hole detection algorithm to accomplish the same. In this case, topological hole information is extracted from the hole detection algorithm, and this information is used as a boundary condition in the dilation stage. The hole detection algorithm would provide the centre, and the radius of the holes and the desired hole radius can be given as a criterion. The radius criterion is only suitable for circular holes. If the geometry also has non-circular holes, hole surface patches from the algorithm can be used to compute the surface area of individual holes. This can be used as a further filtering criterion. Then the faces which are part of the desired holes are not diffused into the



(a) Mode 1



(b) Mode 2

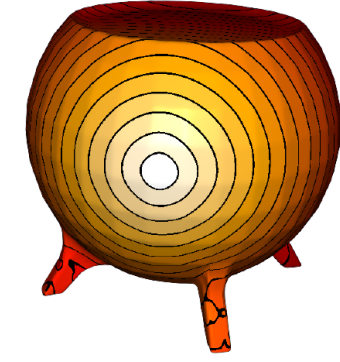


(c) Mode 3

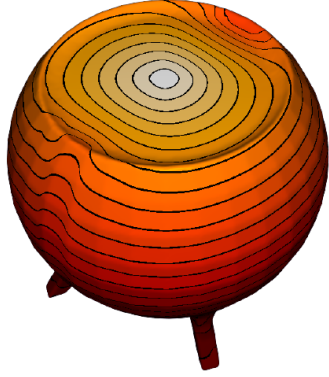
**Figure 18:** First 3 Eigen modes of a shrink wrapped mesh

volume, thus preserving the structure. The algorithm 2 would have to be modified for Explicit Genus control, and it can be seen in algorithm 7

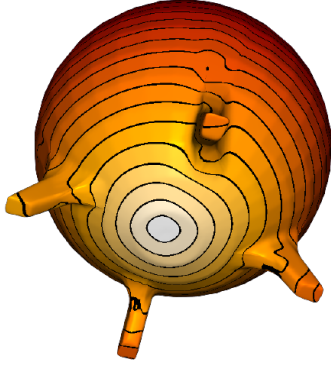
As explained earlier, the dilation process happens layer by layer. Therefore, to achieve selective genus control, one would have to ignore the blacklisted boundary cells for the dilation process.



(a) Side



(b) Top

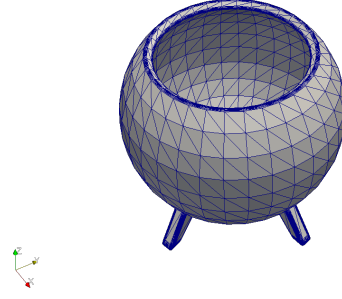


(c) Bottom

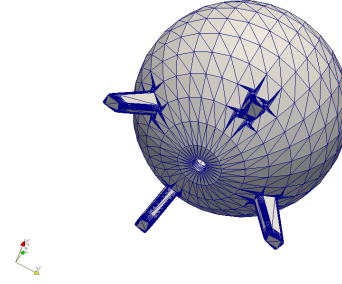
**Figure 19:** Geodesic distance computed on shrink wrapped meshes at different sources using Heat method

### 5.3 Fluid Volume Extraction

Fluid volume extraction is yet another excellent application of our shrink wrapping algorithm. If the goal is to generate the fluid volume or topological holes in a geometry, simple boolean operations help extract these volumes. There are practical difficulties in extracting topological holes in a geometry (multiple holes) since



(a) Bigger hole



(b) Smaller hole

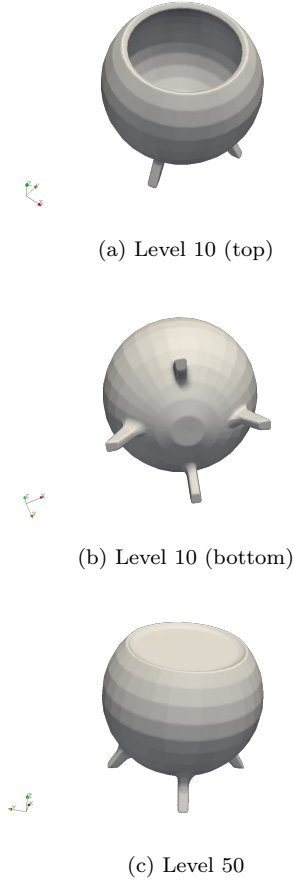
**Figure 20:** An example geometry with a big hole on top and a small hole at the bottom

there will be a lot of noise to sift through. However, suppose the industrial practitioner is interested in extracting a single fluid volume of an interior of a car. In that case, it is possible to automate the process entirely. Once we have a Genus zero shrink-wrapped surface, a straightforward algorithm can be laid out with the following steps

1. Subtract the genus zero shrink wrapped geometry from the input geometry
2. Split the result based on connectivity and compute the component volumes
3. Largest volume geometry is the fluid volume

We did not implement any of the boolean operations for this algorithm and used the existing functionalities from CGAL[16].

Smoothed particle hydrodynamics (SPH) is a meshless method requiring a volumetric point distribution for numerical simulation. In example 1, we show a partial car in figure 22 that has been shrink-wrapped, and its



**Figure 21:** Shrink wrapped geometry for different topological sphere levels

fluid volume has been extracted for a smoother particle hydrodynamic simulation.

In the second example, we show a case for raspberry pi, and its shrink-wrapped geometry and the subsequent fluid volume in figure 23. Again, it can be seen that our algorithm produces a very clean fluid volume. The results can be further de-noised to turn them into developable surfaces.

#### 5.4 External aerodynamic simulation

One of the primary focuses of our investigation is external aerodynamic simulations. Since they do not require all the internal components of a geometry, a simplified geometry can be considered during early prototyping. We ran the RANS simulations on a generic shrink wrapped car geometry using OpenFoam[20]. The car geometry was meshed using the snappy-HexMesh tool from OpenFoam with a base refinement of 10 cells in all three directions. We considered the

---

**Algorithm 7:** Dilation of the input surface (with Genus control)

---

**Result:** Dilated surface stored in the voxelized mesh

```

Detect holes using hole detection algorithm;
Mark the cells that intersect with the holes and
mark them as blacklisted cells;
Initialize interior_cells as seed_cells;
Ensure that the blacklisted cells are removed
from the initial seed_cells;
Initialize current_topological_sphere_level to 0;
while current_level ≤ topological_sphere_level
do
    Initialize a newer_seeds_cells_id vector;
    forall cells in seed_cells do
        forall cell_neighbours in voxelized_mesh do
            if Neighbour is outside cell then
                Add neighbour to
                newer_seeds_cells_id;
            end
        end
    Set newer_seeds_cells_id as seed_cells;
    Increment current_topological_sphere_level;
    if current_topological_sphere_level eq
        topological_sphere_level then
        Store these cells as
        topological_sphere_cells;
    end
end
end

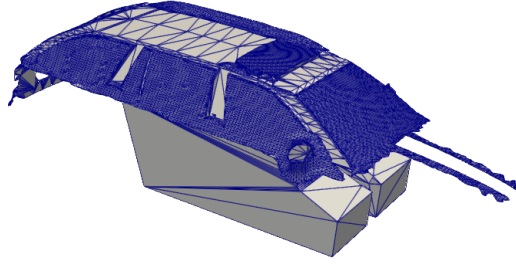
```

---

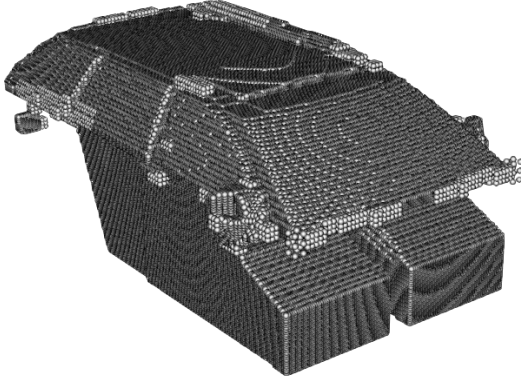
entire car geometry for numerical simulation without using any symmetry boundary conditions. We used a steady-state incompressible SIMPLE solver for solving the Reynolds Averaged Navier Stokes equation with a  $k-\omega$  SST turbulence model. A velocity inlet with a velocity of 20 ms was used as the boundary condition. The shrink-wrapped geometry produces physically consistent results, as shown in the figure 24. We have not made a rigorous mathematical analysis or experimental verification for these simulations. We only performed this simulation to show the suitability of shrink-wrapped geometries for computational fluid dynamic simulations.

## 6. COMPARISON AGAINST SIMILAR APPROACHES

Many recent papers in geometry processing use deep learning-based approaches to solve geometric problems. One such recent article is *Point2Mesh*[21] where the authors shrink wrap an oriented point cloud based on self-similarity. Their algorithm is built on mesh-based convolutional neural networks and similar algorithms found in computer vision. We extended our



(a) Wrapped Car

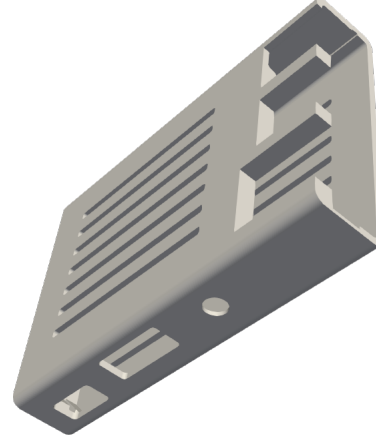


(b) Car fluid volume

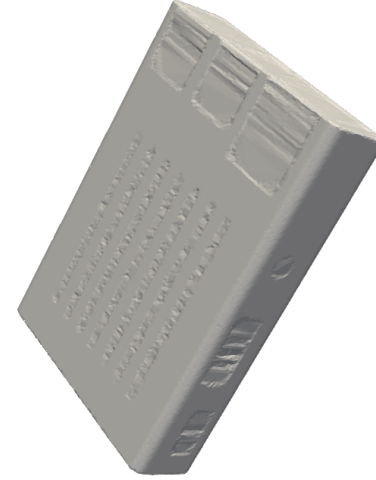
**Figure 22:** A Partial car geometry that has been shrink wrapped and its fluid volume extracted as a volumetric point cloud. Volumetric point clouds are otherwise considered as particle distributions for meshless methods like Smoothed particle hydrodynamics method.

shrink wrapping algorithm for point clouds to make a fair comparison. Since we rely on generalized winding numbers for inside-outside segmentation, there is a straightforward extension to point clouds. Usually, this is done by computing point areas using a Voronoi diagram[12]. However, we found that such a complex approximation is not always required. We compute a series of local triangulations and consistently ensure their orientation using a greedy approach. Hence, our algorithm does not require an oriented point cloud. This modification ensures that we do not have to modify the rest of our shrink wrapping algorithm. Once the inside-outside segmentation is done in the octree, the rest of the algorithm remains the same. We chose the same geometries as the authors, and we found that we produce similar quality results in most cases. While we provided a variety of heuristics to avoid this in a surface mesh-based approach, we did not thoroughly investigate the same for point clouds since it was beyond the scope of our work.

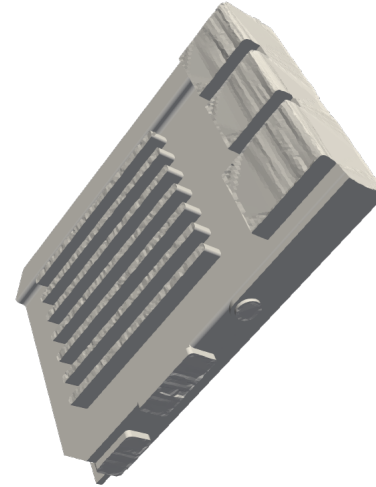
An observation can be made that our algorithm complements the authors' work nicely. If our algorithm is considered an initial priori, it improves the con-



(a) Raspberry pi case geometry



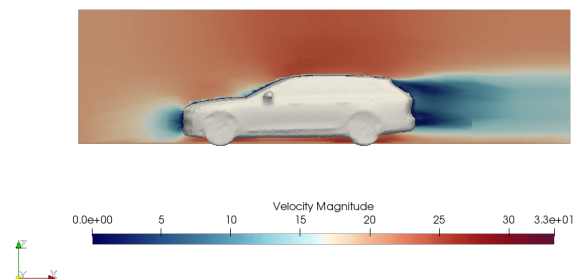
(b) Wrapped Geometry



(c) Fluid volume

**Figure 23:** A raspberry pi case and its fluid volume

vergence speed of *Point2Mesh* algorithm. For exam-



**Figure 24:** External aerodynamic simulation of a generic car model (shrink wrapped)

ple, their algorithm relies on an initial mesh computed based on a convex hull approach and converges to a ground truth based on self-similarity. However, our algorithm’s mesh before the projection stage serves as a better priori. It also converges the *Point2Mesh*[21] algorithm in a fraction of the time. It can also be noticed that their approach is not meant for mechanical parts, and features found in CAD geometries cannot be preserved without significant modifications. Our goal is to produce genus-zero surfaces for aerodynamic simulations. We optionally provide variants that allow various levels of control over the genus of the wrapped surface. However, their approach is strictly a surface construction approach and does not consistently produce zero surfaces. It can only be achieved by stopping the algorithm halfway; the reconstruction might not be accurate globally in such cases, and a projection might be required.

## 7. LIMITATIONS

Since the proposed algorithms are built on top of morphological operators, they inherit the drawbacks of mathematical morphology. In concave regions, the erosion stops once it hits the closest triangle in the mesh. This leads to over closing of specific features in the mesh. However, since the primary application for our algorithms is external aerodynamic simulations, these do not make a massive difference in the macro scale. There are scenarios, however, where the algorithm can significantly alter the geometry. The techniques listed in section 4.3 can alleviate this problem to a large extent. However, no algorithmic guarantees can be made here.

## 8. CONCLUSION

We presented a practical algorithm that can perform Genus simplified shrink wrapping for polyhedral surfaces with the help of morphological operators. We also show that these algorithms extend easily for point clouds. One can implement the algorithms proposed in

this paper in the same mesh used for numerical simulation, thereby avoiding another expensive volumetric mesh generation process. The algorithms also run at a linear runtime and are not heavily CPU bound. Furthermore, user-defined constraints and additional interactivity could lead to further improvements in the output quality of the algorithm. Finally, our fluid volume extraction variant of the algorithm can significantly benefit industrial fluid dynamic practitioners.

## ACKNOWLEDGEMENTS

Österreichische Forschungsförderungsgesellschaft has funded this research under an industrial Ph.D. grant titled “**HIOMESH**”.

## References

- [1] Attene M., Campen M., Kobbelt L. “Polygon Mesh Repairing: An Application Perspective.” *ACM Comput. Surv.*, vol. 45, no. 2, Mar. 2013. URL <https://doi.org/10.1145/2431211.2431214>
- [2] Esteve J., Brunet P., Vinacau A. “Approximation of a Variable Density Cloud of Points by Shrinking a Discrete Membrane.” *Computer Graphics Forum*, vol. 24, no. 4, 791–807, 2005
- [3] Nooruddin F., Turk G. “Simplification and repair of polygonal models using volumetric techniques.” *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, 191–205, 2003
- [4] Lee Y.K., Lim C.K., Ghazialam H., Vardhan H., Eklund E. “Surface Mesh Generation for Dirty Geometries by Shrink Wrapping using Cartesian Grid Approach.” P.P. Pébay, editor, *Proceedings of the 15th International Meshing Roundtable*, pp. 393–410. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006
- [5] Wang Z.J., Srinivasan K. “An adaptive Cartesian grid generation method for ‘Dirty’ geometry.” *International Journal for Numerical Methods in Fluids*, vol. 39, no. 8, 703–717, 2002
- [6] Vijai Kumar S., Vuik C. “A Simple and Fast Hole Detection Algorithm for Triangulated Surfaces.” *Journal of Computing and Information Science in Engineering*, vol. 21, no. 4, 02 2021. URL <https://doi.org/10.1115/1.4049030.044502>
- [7] Najman L., Talbot H. *Introduction to Mathematical Morphology*, chap. 1, pp. 1–33. John Wiley & Sons, Ltd, 2013. URL <https://doi.org/10.1002/9781118600788.ch1>





**Figure 25:** Few point cloud geometries from *Point2Mesh*[21] along with its output. It can be noticed that our wrap algorithm produces equally smooth results except for a few artefacts created as a result of morphological operators.

- [8] Jeulin D. *Analysis and Modeling of 3D Microstructures*, chap. 19, pp. 421–444. John Wiley & Sons, Ltd, 2013. URL <https://doi.org/10.1002/9781118600788.ch19>
- [9] Chen Z., Panozzo D., Dumas J. “Half-Space Power Diagrams and Discrete Surface Offsets.” *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 10, 2970–2981, 2020
- [10] Sellán S., Kesten J., Sheng A.Y., Jacobson A. “Opening and Closing Surfaces.” *ACM Trans. Graph.*, vol. 39, no. 6, Nov. 2020. URL <https://doi.org/10.1145/3414685.3417778>
- [11] Gärtner B. “Fast and Robust Smallest Enclosing Balls.” *Proceedings of the 7th Annual European Symposium on Algorithms*, ESA ’99, pp. 325–338. Springer-Verlag, London, UK, UK, 1999
- [12] Barill G., Dickson N., Schmidt R., Levin D.I., Jacobson A. “Fast Winding Numbers for Soups and Clouds.” *ACM Transactions on Graphics*, 2018
- [13] Liu S., Wang C.C.L. “Fast Intersection-Free Offset Surface Generation From Freeform Models With Triangular Meshes.” *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 2, 347–360, 2011
- [14] Lorensen W.E., Cline H.E. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm.” *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’87, p. 163–169. Association for Computing Machinery, New York, NY, USA, 1987
- [15] Blanco J.L., Rai P.K. “nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) with KD-trees.” <https://github.com/jlblancoc/nanoflann>, 2014
- [16] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.3 edn., 2021. URL <https://doc.cgal.org/5.3/Manual/packages.html>



- [17] Alexa M., Wardetzky M. “Discrete Laplacians on General Polygonal Meshes.” *ACM Trans. Graph.*, vol. 30, no. 4, Jul. 2011. URL <https://doi.org/10.1145/2010324.1964997>
- [18] Vallet B., Lévy B. “Spectral Geometry Processing with Manifold Harmonics.” *Computer Graphics Forum*, vol. 27, no. 2, 251–260, 2008. URL <https://doi.org/b24vx4>
- [19] Crane K., Weischedel C., Wardetzky M. “The Heat Method for Distance Computation.” *Commun. ACM*, vol. 60, no. 11, 90–99, Oct. 2017. URL <http://doi.acm.org/10.1145/3131280>
- [20] Foundation T.O. “OpenFOAM v8 User Guide.”, 2021. URL <https://cfd.direct/openfoam/user-guide>
- [21] Hanocka R., Metzer G., Giryas R., Cohen-Or D. “Point2Mesh: A Self-Prior for Deformable Meshes.” *ACM Trans. Graph.*, vol. 39, no. 4, jul 2020. URL <https://doi.org/10.1145/3386569.3392415>
- [22] Jacobson A., Kavan L., Sorkine-Hornung O. “Robust Inside-Outside Segmentation Using Generalized Winding Numbers.” *ACM Trans. Graph.*, vol. 32, no. 4, Jul. 2013
- [23] Carrier J., Greengard L., Rokhlin V. “A Fast Adaptive Multipole Algorithm for Particle Simulations.” *SIAM J. Sci. Stat. Comput.*, vol. 9, no. 4, 669–686, Jul. 1988. URL <https://doi.org/10.1137/0909044>

## APPENDIX A: GENERALIZED WINDING NUMBER BASED SOLID ANGLE FOR SURFACE SEGMENTATION

We rely on a classical differential geometry idea called winding numbers, which uses solid angles for surface segmentation. For a given surface  $S$ , for a query point  $p$ , the solid angle is the signed surface area of the projection of  $S$  onto the unit sphere centred at  $p$  as shown in figure 26. We rely on its definition in discrete setting[22].

$$\omega(\mathbf{p}) = 2 * \tan^{-1} \left( \frac{\det([\mathbf{abc}])}{abc + (\mathbf{a} \cdot \mathbf{b})c + (\mathbf{b} \cdot \mathbf{c})a + (\mathbf{c} \cdot \mathbf{a})b} \right)$$

$$\text{Triangle} = \{v_i, v_j, v_k\}$$

$$\mathbf{a} = v_i - p, \mathbf{b} = v_j - p, \mathbf{c} = v_k - p$$

$$a = \|\mathbf{a}\|$$

$$b = \|\mathbf{b}\|$$

$$c = \|\mathbf{c}\|$$

(4)

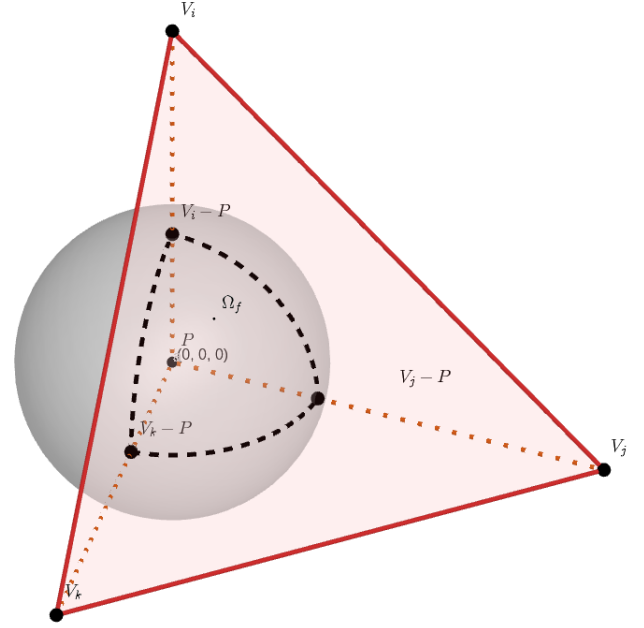


Figure 26: Solid angle of a query point

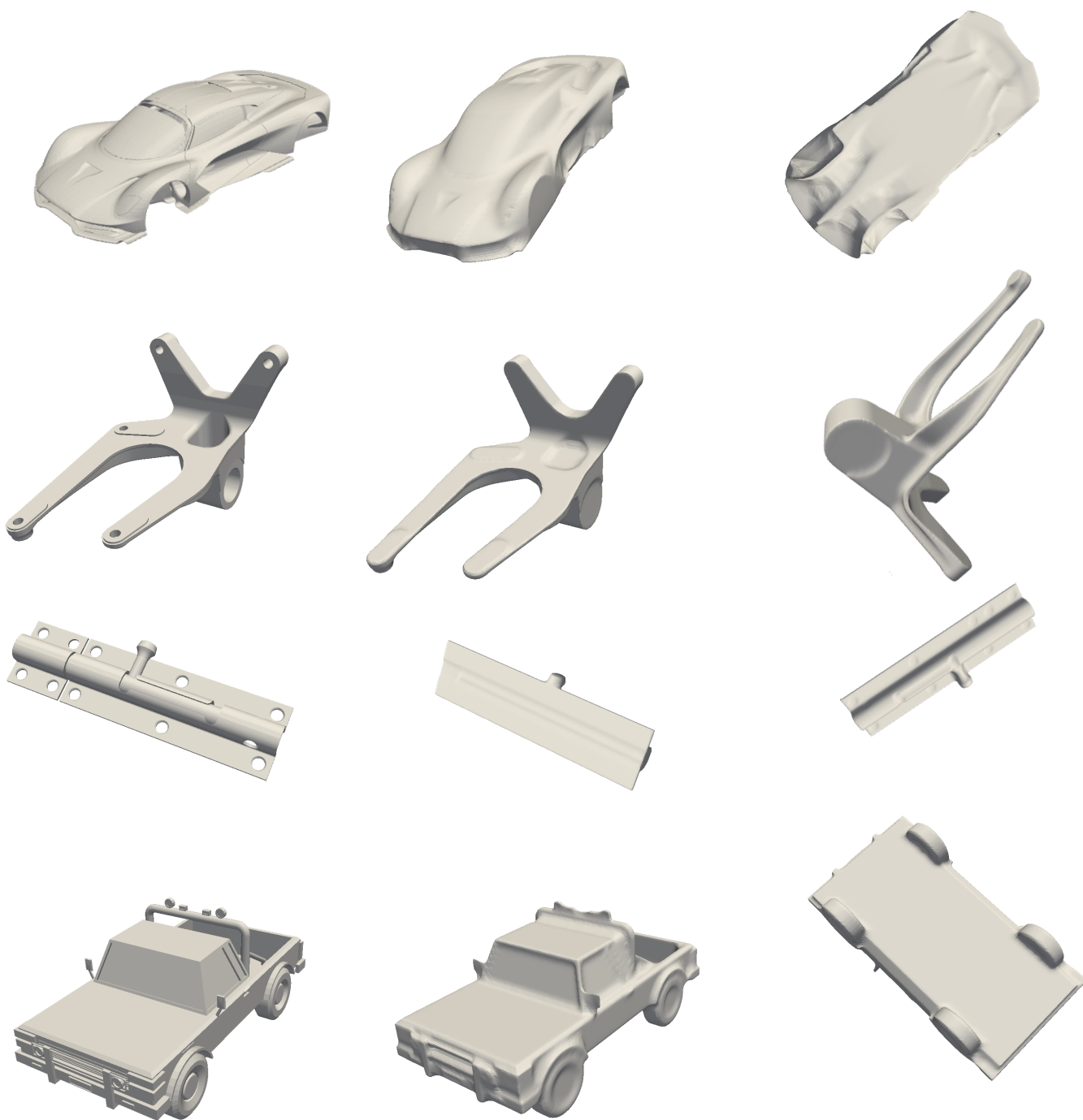
Given this relation for solid angle given by  $\omega(\mathbf{p})$ , we can compute winding number as follows

$$w(\mathbf{p}) = \sum_{n=1}^{n_{Triangles}} \frac{1}{4\pi} \omega_f(\mathbf{p})$$

For every query point, the direct implementation  $w(\mathbf{p})$  would require the contribution of all triangles in the surface mesh. Since this would yield a solution with time complexity of  $O(n^2)$ , we rely on the work of Gavin Barill et al[12]. They proposed a fast multipole method[23] style implementation that uses direct computation for triangles near the query point and approximates the result everywhere else, making it a  $O(\log(n))$  algorithm.

## APPENDIX B: VARIOUS INPUT GEOMETRIES AND THEIR SHRINK WRAPPED RESULTS

We show a few geometries and their shrink-wrapped results. The results shown below are shrink-wrapped with a topological sphere level of 15. In all the cases shown below, our algorithm could produce a genus zero surface consistently. It also creates a two-manifold mesh suitable for external aerodynamics or finite element analysis simulations.



**Figure 27:** A variety of geometries and their shrink wrapped results.

# SMOOTHING OF SHELL MESHES ON FACETED B-REP GEOMETRY

Harold J. Fogg<sup>1</sup>

Jonathan E. Makem<sup>1</sup>

<sup>1</sup>*Meshing & Abstraction, Simulation and Test Solutions, Siemens Digital Industry Software, 112 Hills Road, Cambridge, UK. jonathan.makem@siemens.com*

## ABSTRACT

Most smoothing methods are designed to move nodes in the interior of a domain whilst holding the nodes on boundaries fixed. By incorporating a stage to move nodes along edges a further improvement in mesh quality may be achieved. This is particularly true of meshes on complex B-rep models with many thin faces and curved edges. The work presented here describes a two-stage process for smoothing shell meshes on the edges and faces of a faceted B-rep geometry. The first stage involves the global smoothing of the mesh to generally improve its quality by Laplacian smoothing along edges with tangent line constraints and variational smoothing on faces. The second local optimisation smoothing stage is designed to make additional local adjustments to the mesh by targeting a specific element quality metric subject to constraints. Again, nodes on edges as well as on faces are smoothed. The approach is demonstrated on a selection of geometries of varying complexity.

**Keywords:** mesh smoothing, mesh optimisation

## 1. INTRODUCTION

On models of industrial complexity even the most advanced mesh generators will produce imperfect initial meshes that require post processing before they are fit for purpose. In order for the particular simulation to be performed efficiently with fidelity to the underlying physics and with the appropriate accuracy the mesh must fulfil certain quality criteria. Unfortunately, the quality criteria are problem dependent and an acceptable mesh for one simulation may be inadequate for another. Certain criteria are almost universally required, such as non-negative element Jacobians, and others are treated as “good rules of thumb”. Typically, element quality metrics such as skew, aspect-ratio, taper, warp, distortion etc. (see e.g. [1]) are required to fall within a specified range.

There are two broad categories of approach for mesh quality improvement: 1) Topological improvement methods such as edge swap, element merge, etc. 2) Smoothing where nodes are repositioned. Both are needed in practice. This paper presents smoothing methods in the second category. Two smoothing methods are outlined, the first to efficiently move all the nodes of the mesh to improve its general quality

overall and the second to make slight local adjustments that improve specific element quality metrics which are difficult to target globally.

## 2. RELATED WORK

Ruiz-Girones et al. [2] presented a hierarchical iterative approach whereby a two stage smoothing and untangling procedure is used to move interior nodes as well as boundary nodes. The technique is applied to analytic CAD geometries and cannot be directly applied to polygonalised faceted representations of geometry. The Gauss-Seidel scheme that the authors use for computing and updating the position of the new node locations may be slow to converge on certain geometries where nodal perturbation is substantial. The single objective function that the authors use will not improve element quality measures which are not strongly correlated with mesh distortion such as Taper for example. The models used to demonstrate the effectiveness of the approach are relatively basic in comparison to the models shown in this work (number of edges and faces  $\sim 100$  versus  $\sim 1k$ ). An overall running time in the order of minutes quoted by the authors suggests that the technique is computationally

expensive.

Garimella et al. [3, 4] reported a procedure to improve the quality of complex polygonal surface meshes on faceted geometries. Again an iterative approach is used and the mesh vertices are repositioned using a non-linear optimisation process. More precisely, two methods are used in the optimisation procedure. The first approach minimises a global condition number and the second method uses a Reference Jacobian Matrix (RJM) to improve mesh quality whilst keeping nodal perturbation to a minimum. The authors don't report exact computation times but instead compare one approach against the other to evaluate computational efficiency as a percentage. All the geometries used are quite simple closed surfaces (although complex for the time) and in some cases the results do not honour the boundary edges of the geometry.

Shivanna et al. [5] proposed a parametrisation and projection-based technique for optimisation of quad shell meshes on underlying triangulated surfaces. The main limitation of the approach is that the nodal perturbation is restricted to surfaces with no infrastructure in place to support node movement along boundaries. The authors use an iterative scheme to update the nodal position on a node-by-node basis. The computational efficiency of the approach has not been evaluated.

Escobar et al. [6] described a Gauss-Seidel approach for tri mesh quality improvement by minimising an objective function derived from algebraic quality measures of the local mesh in the immediate vicinity of the node being perturbed. Constraints are imposed to ensure the objective function restricts the node within a feasible region. The authors construct a local parameter space via orthographic projection but this is not ideal in the vicinity of  $G^1$  discontinuities. The approach is not applied to quad meshes and nodal perturbation along edges is not possible. The effectiveness of the method on complex geometries with many faces and edges is not demonstrated.

Gargallo-Peiro et al. [7] developed a continuous optimization procedure to improve the quality of meshes on parametrised CAD surfaces by smoothing and untangling. Initially, the optimisation process did not consider the prescribed element size so the authors also used the size-shape distortion measure that combines the previous distortion measures to produce a mesh that preserves a prescribed element size field and generates well-shaped elements. Further work is required to extend the untangling capability on such meshes. The approach isn't extended to smooth nodes on edges.

The work presented here outlines a practical method for smoothing large shell meshes on faceted B-rep

models of industrial complexity, unlike other methods which are demonstrated on simpler CAD models. The smoothing of nodes on piece-wise linear geometry edges is achieved without using parametrisation derivatives which are not necessarily available. A complementary local optimisation smoothing method is also described which can be applied to nodes on geometry faces and edges to target specific element quality metrics. These methods are straightforward to implement and have reasonable execution times when run on large meshes of complex models.

### 3. FACETED B-REP GEOMETRY

In 3D solid modelling or CAE software such as Simcenter 3D [8] a boundary representation (B-rep) of the modelling geometry is used. This comprises topological components (faces, edges and vertices) and the connections between them, along with geometric definitions for those components (surfaces, curves and points, respectively). The geometric definitions may be continuous mathematical equations (e.g. splines, NURBS) or facetings where simply connected triangles represent geometry faces and geometry edges are described by polylines composed of triangle edges. In Simcenter 3D the modelling geometry is always converted to a faceted B-rep known as a 'polygon geometry' which is then simplified and de-featured using merging operations to eliminate artefacts and details below the mesh size. The mesh is then generated on the polygon geometry. The continuous geometry to which the polygon geometry approximates is inferred when needed by using a variety of numerical techniques including triangular Bezier patches and Hermite interpolation.

The majority of shell meshing algorithms essentially work in 2D and to apply them to 3D faces a bijective parametrisation is required. For faceted geometry faces a range of practical and robust methods are available [9, 10].

### 4. GLOBAL SMOOTHING

The global smoothing procedure consists of five steps as shown in Fig. 1. These are iterated over until the solution converges or the number of iterations reaches a prescribed maximum number. Pseudo-code for the procedure is given in algorithm 1.

Algorithm 1: Global smoothing pseudocode

```

for i in range (num iterations):
    active edges = edges not converged
    smooth active edges

    active faces = faces not converged
    for face in active faces:
        smooth face

```

Geometry edges and faces are marked as converged if

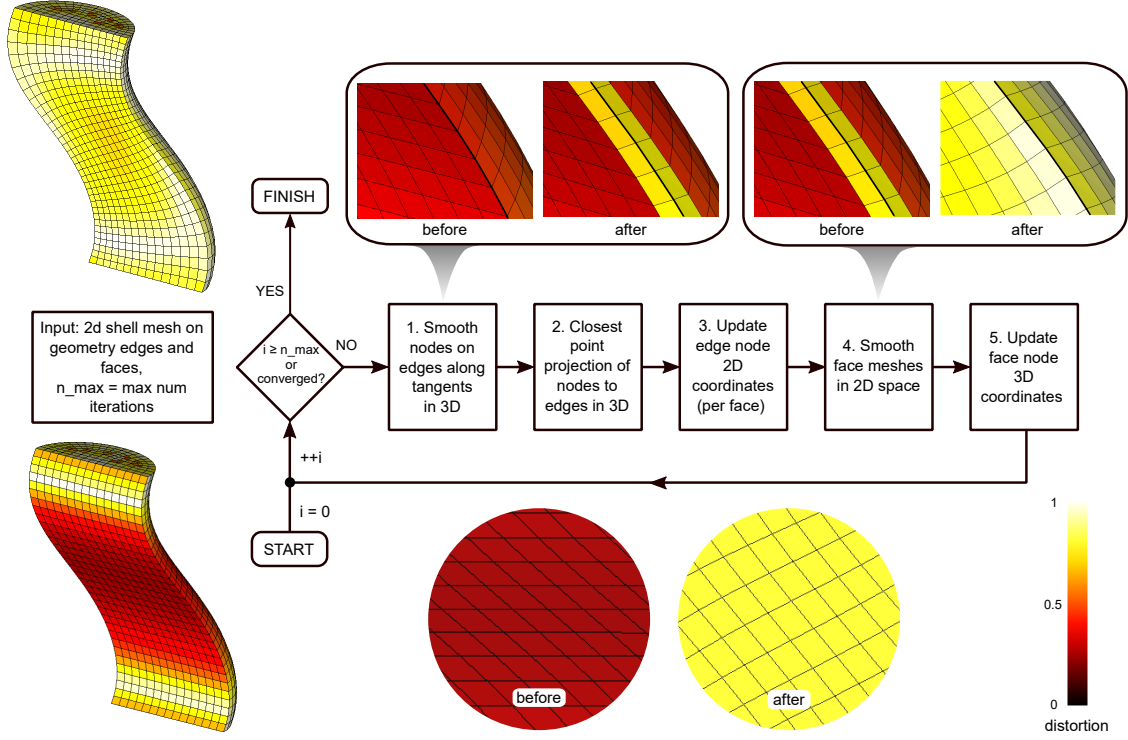


Figure 1: Global smoothing procedure

the maximum movement of a node in the previous iteration is below a small percentage of the local element size (e.g. 4%).

In Fig. 1 the meshes are coloured by element distortion [11], a general element quality metric for triangles and quadrilaterals where ideal elements have a value of 1 and degenerate elements a value of 0.

#### 4.1 Smooth nodes on geometry edges along tangents in 3D

Laplacian smoothing is used with tangent constraints to smooth the nodes along a geometry edge. The unique solution is the distribution of node positions,  $\mathbf{x}$ , that minimises the sum of squared distances between neighbouring nodes (*i.e.* those that are connected by element edges),

$$\begin{aligned} \operatorname{argmin}_{\{\mathbf{x}_i\}} \sum_{i=0}^{\#\text{mesh nodes}} e_i, \\ e_i = \sum_{j=0}^{\#\text{neighbours}_i} \frac{\alpha_{ij}}{2} \|\mathbf{x}_j - \mathbf{x}_i\|^2 + \frac{\beta_i}{2} \|\mathbf{x}_i - \mathbf{x}_{i0}\|^2. \end{aligned} \quad (1)$$

The coefficient  $\alpha_{ij}$  can be chosen to prioritise certain edges and also to reduce the dominance of long

edges by setting the values as inversely proportional to the initial edge length squared. The second term of Eqn. (1) with coefficient  $\beta_i$  (e.g. =0.0005) is added to to penalise large displacements from the initial positions,  $\mathbf{x}_{i0}$ . This is a regularisation strategy. Since the initial positions are reset in each iteration nodes may move a long way from their pre-smoothing positions after a few iterations. Adding constraints to keep nodes on curved geometry edges would make the problem non-linear. Instead tangent line constraints are added to keep the nodes close to the geometry edges, but not exactly on them. The nodes are projected back to the edge in the subsequent step.

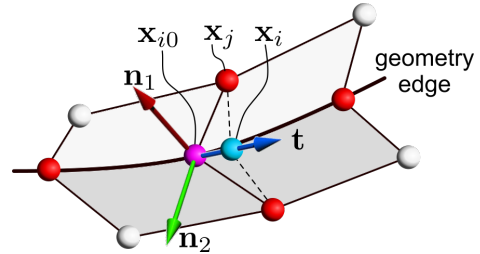
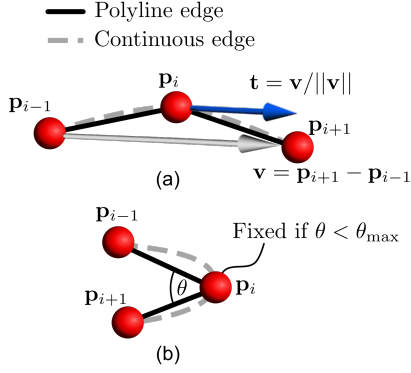


Figure 2: Laplacian smoothing of node on geometry edge

Tangents are approximated by the unitised displacements from the previous node to the next node, as show in Fig. 3 (a). The tangents that are computed are thus unaffected by small shape details of the ge-

ometry edge that are not captured by the initial mesh discretisation.



**Figure 3:** Approximating tangent vectors

If the angle between the previous node, the central node and the next node,  $\theta$ , is below a tolerance,  $\theta_{\max}$ , then the node is fixed and not smoothed. Without doing this there is a risk that the local tangent approximation of the geometry edge is too crude and problems with mesh tangling could occur during the next projection stage. Experiments have found that using  $\theta_{\max} = 80^\circ$  avoids such issues.

For a node on a geometry edge, as shown in Fig. 2, at initial position  $\mathbf{x}_{i0}$  with tangent vector  $\mathbf{t}$ , Laplacian smoothing with a tangent constraint satisfies

$$\operatorname{argmin}_{\mathbf{x}_i} e_i \quad (2)$$

$$\text{s.t. } g_{i1} = (\mathbf{x}_i - \mathbf{x}_{i0}) \cdot \mathbf{n}_1 = 0, \quad (3)$$

$$g_{i2} = (\mathbf{x}_i - \mathbf{x}_{i0}) \cdot \mathbf{n}_2 = 0, \quad (4)$$

where  $\mathbf{n}_1$  and  $\mathbf{n}_2$  are two mutually perpendicular unit vectors are perpendicular to the edge tangent vector  $\mathbf{t}$ . A possible choice for  $\mathbf{n}_1$  and  $\mathbf{n}_2$  are the the normal and bi-normal in a Frenet-Serret frame. Using Lagrange multipliers the Lagrangian is

$$L_i(\mathbf{x}_i, \lambda_1, \lambda_2) = e_i(\mathbf{x}_i) - \lambda_1 g_{i1}(\mathbf{x}_i) - \lambda_2 g_{i2}(\mathbf{x}_i). \quad (5)$$

The solution is an extremum,

$$\nabla_{\mathbf{x}_i, \lambda_1, \lambda_2} L_i = 0 \quad (6)$$

$$\Rightarrow \begin{cases} \frac{\partial L_i}{\partial \mathbf{x}_i} = \frac{\partial e_i}{\partial \mathbf{x}_i} - \lambda_1 \frac{\partial g_{i1}}{\partial \mathbf{x}_i} - \lambda_2 \frac{\partial g_{i2}}{\partial \mathbf{x}_i} = 0, \\ \frac{\partial L_i}{\partial \lambda_i} = -g_i = 0. \end{cases} \quad (7)$$

The equation  $\frac{\partial L_i}{\partial \mathbf{x}_i} = 0$  can be expanded and simplified to

$$\begin{aligned} \sum_{j=0}^{\# \text{neighbours}_i} (\alpha_{ij} + \beta_i) \mathbf{x}_i - \sum_{j=0}^{\# \text{neighbours}_i} \alpha_{ij} \mathbf{x}_j \\ + \lambda_1 \mathbf{n}_1 + \lambda_2 \mathbf{n}_2 = \beta_i \mathbf{x}_{i0}. \end{aligned} \quad (8)$$

The nodes on edges may be smoothed one at a time in a local iteration scheme or alternatively the nodes

on edges can be smoothed together in one shot in a global linear system. Equations (3), (4) and (8) give a linear system of 5 equations with 5 unknowns  $(x_i, y_i, z_i, \lambda_{i1}, \lambda_{i2})$  for a single node. The nodal contributions can be assembled together into a global linear system for the entire mesh and decomposed by Schur complement to give a sparse system of dimension 5 times the number of free nodes. For a row corresponding to a free node the number of non-zero entries is three times the number of free nodes that are adjacent to the node.

## 4.2 Closest point projection of nodes to geometry-edges

A simple closest point algorithm is used to project the smoothed edge nodes back onto the polyline geometry edge. This algorithm has quadratic complexity so it becomes prohibitively expensive if the polyline has many points. If the polyline has more points than  $O(10^2)$  then spatial trees (e.g. R-trees) can be used to reduce the expense of the algorithm.

After the global smoothing has been completed the nodes are lifted off the polyline geometry edges and repositioned on the inferred continuous geometry edge by cubic Hermite interpolation.

## 4.3 Update geometry edge node 2D positions (per face)

Since the 3D positions of the nodes on the geometry edges have been moved in steps 1 and 2 the 2D positions have to be updated. This can be done efficiently using the closest point projection data of step 2. The closest polyline segment has been identified and the 2D positions of the closest point along the segment can be calculated by linearly interpolating between the 2D positions of the facet vertices at the start and end of the segment. For every node on an edge its 2D positions of every connected face (1 for a free edge, 2 for a manifold edge, +2 for non-manifold connections) are updated. On seam edges (see e.g. Fig. 9) nodes have two corresponding 2D positions and both must be updated.

## 4.4 Smooth face meshes in 2D space

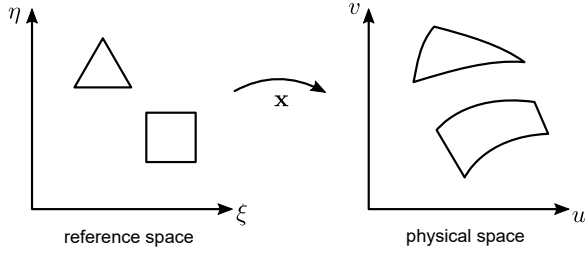
In response to the updated positions of the nodes on edges the interior nodes on the faces are smoothed with the edge nodes held fixed. All the interior nodes may be smoothed or to improve computational efficiency a subset of nodes in proximity to edges can be smoothed. The subset can be defined based on:

- adjacency: a specified number of layers from the edges (0 – no face smoothing, 1 – the neighbours



- of the edge nodes, 2 – the neighbours of neighbours of the edge nodes, etc.), or
- distance: the nodes within an offset distance from the edges.

For the smoothing of the mesh in 2D the variational smoothing scheme of Knupp [12, 13] is used. It is assumed that there is a twice differentiable map,  $\mathbf{x}$ , for every local region of the mesh to go from 2D reference space  $(\xi, \eta)$  to 2D physical space  $(u, v)$  (which is a face's parametric space). The element shapes in



**Figure 4:** Reference to physical space map

reference space are ideal and unit in length, e.g. equilateral triangles and squares. A variational principle is a rule for evaluating an entire mesh to give a single real number. A smoothed mesh minimises some variation principle, typically of the form

$$I[\mathbf{x}] = \iint H d\xi d\eta. \quad (9)$$

The metric tensor,  $G$ , of the map from reference space to physical space and it governs the shapes of the elements in physical space,

$$G = \begin{bmatrix} g_{11} & g_{12} \\ g_{12} & g_{22} \end{bmatrix} \quad (10)$$

$$g_{11} = \mathbf{x}_\xi \cdot \mathbf{x}_\xi \quad (11)$$

$$g_{12} = \mathbf{x}_\xi \cdot \mathbf{x}_\eta \quad (12)$$

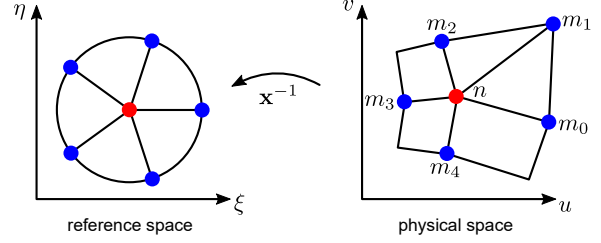
$$g_{22} = \mathbf{x}_\eta \cdot \mathbf{x}_\eta \quad (13)$$

Of practical interest are variational principles that are expressed in terms of the metric tensor and relate to geometric properties. Some important integrands are listed in Tab. 1.

Principle	$H$
Length Squared	$\text{tr}G$
Area Squared	$\det G$
Winslow	$\frac{\text{tr}G}{\det G}$
Orthogonality	$g_{12}^2$
Combinations	$\sum_i \alpha_i H_i$

**Table 1:** Some metric tensor based variational principle integrands

The inverse map  $\mathbf{x}^{-1}$  at a node  $n$  is assumed to equally spread the neighbouring nodes  $m_i$  around a unit circle as shown in Fig. 5. Finite difference stencils are



**Figure 5:** Reference to physical space inverse map

derived for approximating the first and second derivatives of the map,  $\mathbf{x}_u$ ,  $\mathbf{x}_v$ ,  $\mathbf{x}_{uu}$ ,  $\mathbf{x}_{uv}$  and  $\mathbf{x}_{vv}$ , by linear expressions using the positions of the neighbouring nodes. For the case of a node with four adjacent elements the diagonal nodes of adjacent quad elements are also used in the stencil.

As described in [13], the Euler-Lagrange equation which minimises the variational principle can be expressed in the form

$$T_{11}\mathbf{x}_{uu} + T_{12}\mathbf{x}_{uv} + T_{22}\mathbf{x}_{vv} = 0. \quad (14)$$

$T_{11}$ ,  $T_{12}$  and  $T_{22}$  are  $2 \times 2$  matrices with entries that only involve  $\mathbf{x}_u$  and  $\mathbf{x}_v$  and partial derivatives of  $H$  with respect to  $g_{11}$ ,  $g_{12}$ ,  $g_{22}$  and  $\det G$ . The method of Picard iterations can be used to solve this non-linear system by treating  $T_{11}$ ,  $T_{12}$  and  $T_{22}$  as constants to give a linearised system of dimension  $2 \times 2$ . This is solved and followed by an update of the matrices  $T_{11}$ ,  $T_{12}$  and  $T_{22}$ . This is repeated for a few iterations until the solution converges.

The node contributions of Eqn. (14) can be assembled into a global system for a face. The dimension would be the number of free nodes if the variational principle is uncoupled with respect to the  $u$  and  $v$  coordinates (Length Squared and Winslow) and twice that otherwise. For a row corresponding to a particular free node the number of non-zero entries is the number of free nodes that are adjacent to the node for an uncoupled variational principle and twice that otherwise.

Combining variational principles is useful to achieve a compromise between the properties controlled by each. No ideal combination exists in general so empirical testing is required to find effective weights for the kinds of meshes that are expected. It has been found that for quad-dominant body panel meshes an effective combination of variational principles is Length Squared, Area Squared and Orthogonality with the weights 1.0, 0.01 and 0.01 at a reference length of 100. Using a dominant weight for Length Squared means that the numerical scheme is stable, whilst the small weight for area prevents the mesh inverting and the small weight for Orthogonality improves the element shapes without causing instability.

## 4.5 Update face node 3D positions

The 3D positions of the face nodes that were smoothed in the previous step must be updated. The bijective parametrisation is stored simply as 2D positions at every facet vertex. The standard operation for mapping a point from 2D to 3D involves first finding the containing or closest facet and then using the three 3D positions of its vertices to interpolate the 3D point. The first task can be done efficiently using bounding boxes and spatial hashing.

## 4.6 Results

Example results of the global smoother are shown in Figs. 9, 7 and 6. Their timing data is given in Tab. 2 where the following notation is used:

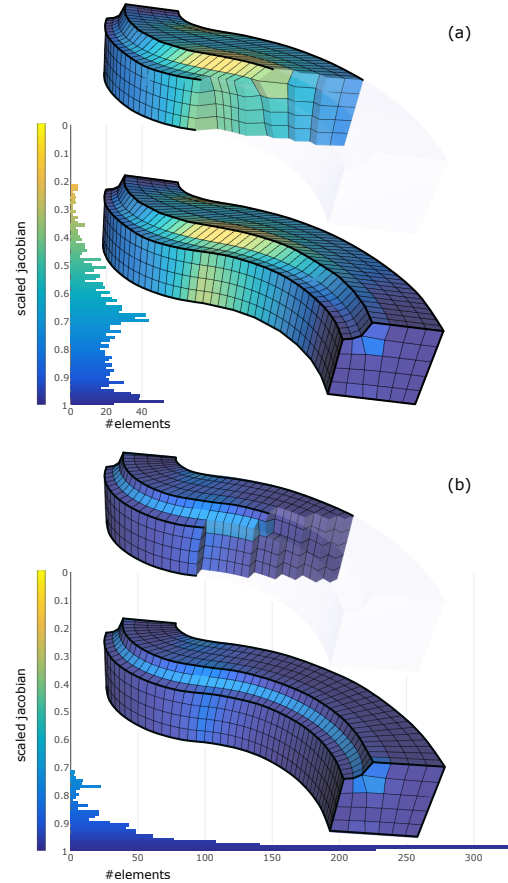
- $n_{\text{edge/face}}$  – the number of nodes on edges/faces that were smoothed.
- $t_{\text{init}}$  – the time to initialise data structures.
- $t_{\text{edge/face}}$  – the time taken to smooth the nodes on edges.
- $t_{\text{proj}}$  – the time taken to do all node projections on edges.
- $t_{\text{total}}$  – the total time taken.

All results were generated using a desktop machine with a Intel Xeon CPU E3-1240 v6 at 3.70GHz (8 CPUs) with 64GB RAM. The algorithms are executed in serial.

A swept hex mesh example is shown in Fig. 6. The wall-faces (i.e the four-sided faces connecting the source and target faces) are transfinite mapped meshed which results in a high degree of distortion in this case. By applying the global smoother to the wall-face shell meshes before the generation of the volume mesh the quality of the final hex mesh is greatly improved.

In Fig. 7 the global smoother is applied to a quad dominant mesh on an Opel body panel [15] that was generated in Simcenter 3D for a transient dynamic crash simulation. Elements in some thin fillet regions which have been mapped meshed exhibit large distortion in the initial mesh. This is significantly alleviated by smoothing. Similarly, the skewed elements in the series of concentric annulus faces around an inner hole are effectively straightened after smoothing.

Fig. 9 shows an example of smoothing a quad mesh on a curved pipe face. The pipe face was constructed by revolving a circular profile curve around two orthogonal axes. A seam edge connecting the edge loops was automatically added in the CAE software to facilitate the parametrisation of the face. Each node along the seam has two 2D positions and both must be smoothed and updated. Global smoothing significantly improves the mesh quality and in this case a



**Figure 6:** Swept hex mesh without (a) and with (b) smoothing the wall face meshes. Mesh sections are shown on top. (Images generated using Hexalab [14].)

global system scheme is found to be more effective than a local iteration scheme. This will be further discussed in the next section.

### 4.6.1 Local iteration versus global system

There are two possible schemes for performing the global smoothing of the mesh:

1. Local iteration – Sequentially visit each free node and solve its new position treating the adjacent nodes as fixed. If the node position is updated immediately it is a *Gauss-Seidel* scheme.
2. Global system – Solve a global system (iteratively if it is non-linear) for the positions of all free nodes together in one shot.

The global system can be solved using a sparse system solver, e.g. [16]. In our implementation a SuperLU direct solver is used for matrices with less than  $10^6$  entries. For larger matrices the memory usage may

model	#edges	#faces	$n_{\text{edge}}$	$n_{\text{face}}$	$t_{\text{init}}$	$t_{\text{edge}}$	$t_{\text{face}}$	$t_{\text{proj}}$	$t_{\text{total}}$
<b>S Offset</b> (Local iteration) (Fig. 1)	6	4	136	1260	0.19s	0.32s	1.01s	0.08s	1.61s
<b>Sweep</b> (Local iteration) (Fig. 6)	15	6	247	782	0.16s	0.28s	0.47s	0.14s	1.06s
<b>Body panel 1</b> (Figs. 7 and 8 (a))	3138	1140	22120	39472					
- Local iteration					12.29s	9.78s	5.87s	0.26s	28.20s
- Global system					12.32s	4.18s	5.73s	0.42s	22.65s
<b>Body panel 2</b> (Fig. 8 (b))	1170	423	8912	16308					
- Local iteration					4.64s	3.37s	2.33s	0.12s	10.47s
- Global system					4.88s	1.25s	1.57s	0.14s	7.84s
<b>Body panel 3</b> (Fig. 8 (c))	2118	789	16245	28070					
- Local iteration					8.19s	6.15s	3.57s	0.23s	18.13s
- Global system					8.70s	2.38s	1.63s	0.30s	13.01s
<b>Pipe</b> (Fig. 9)	3	1	121	1452					
- Local iteration					0.19s	0.07s	0.77s	0.01s	1.03s
- Global system					0.20s	0.12s	1.43s	0.02s	1.77s

Table 2: Timings for global smoothing.

approach the heap memory limits and an indirect bi-conjugate gradient stabilized solver is used which is up to ten times slower.

The two main advantages of the local iteration Gauss-Seidel scheme are the ease of its implementation and that it is straightforward to ensure that no element quality regressions (e.g. negative Jacobian) are caused by smoothing. This is done by performing checks after a local smoothing iteration and avoiding updating the position if necessary. On the other hand the global system scheme may be faster. In Fig. 8 (top) the convergence behaviour of global smoothing using both the local iteration and global system schemes on three large meshes ( $\sim 10k$ ) on body panel geometries with many faces and edges ( $\sim 1k$ ) are shown. Both schemes reach convergence in less than 7 global iterations. In Fig. 8 (bottom) it can be seen that the distortion quality improvements of the mesh are similar. However, as reported in Tab. 2 the execution times are marginally faster for the global system scheme.

For cases where smoothing must move the nodes relatively far from their initial positions to reach convergence the global system scheme may outperform the local iteration scheme. For example in Fig. 9 a quad mesh on a pipe face is smoothed to convergence using a local iteration scheme and a global system scheme. A comparison of the histograms for element distortion indicates that the local iteration scheme converged prematurely and that a superior result was produced by using the global scheme.

Overall for most B-rep models resembling body panels with many edges and faces the advantages of the local iteration scheme outweigh those of the global iteration scheme. The ability to maintain valid-in valid-out meshes at every stage of the smoothing process is of major benefit to the algorithm robustness in the general case.

## 5. LOCAL OPTIMISATION SMOOTHING

The global smoothing procedure is capable of large-scale changes to the mesh to improve its general quality. It can do this robustly and quite efficiently. However, some element quality metrics might be important to the analyst but they may not be expressible in the form a variational principle in Eqn. (9). Moreover, they might not be strongly correlated with the chosen variational principle that is used for global smoothing. To deal with this problem another phase of smoothing is applied to the mesh.

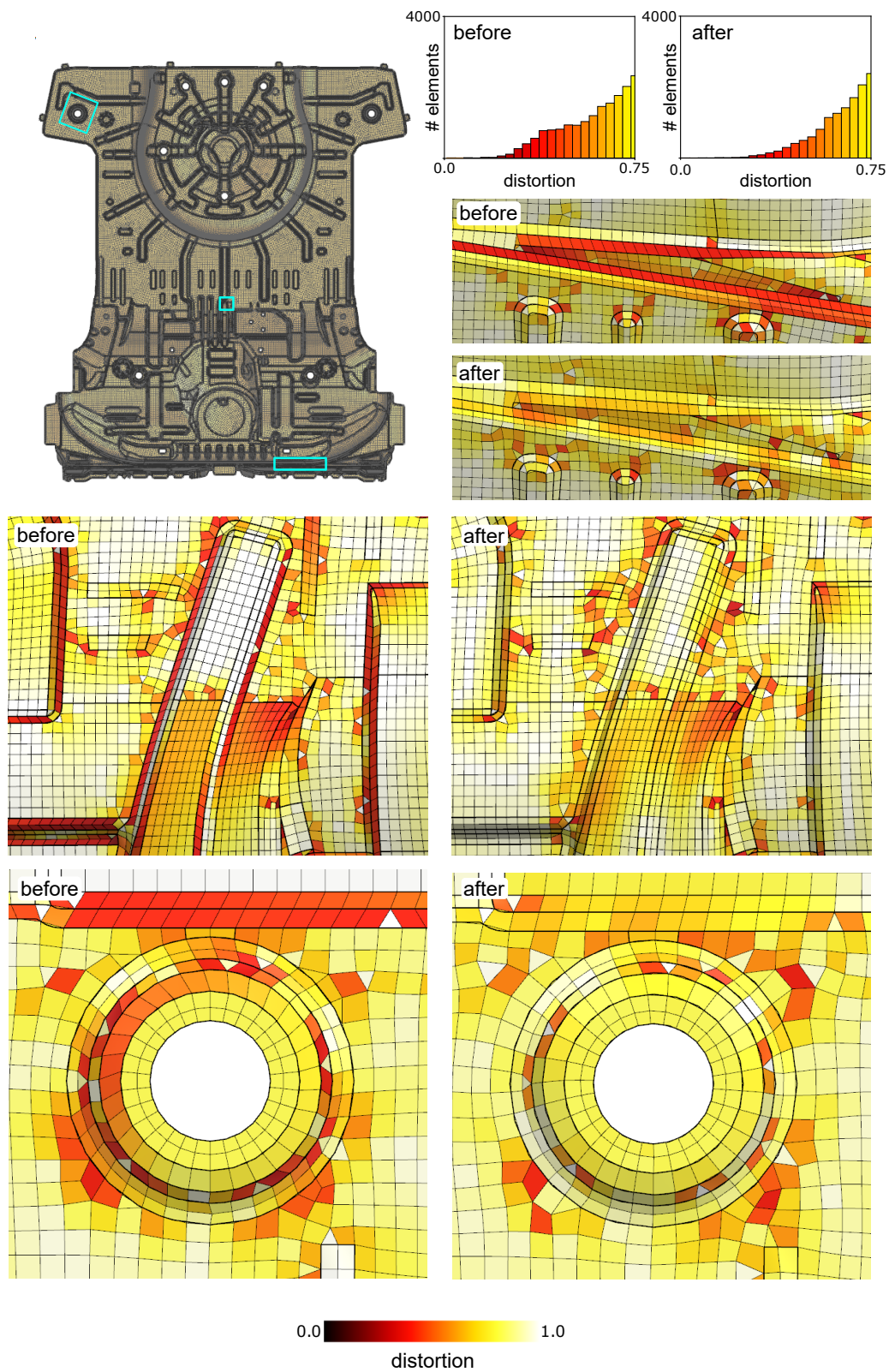
Local optimisation smoothing is designed to solve the following problem: For a *target* element quality metric and a specified range, smooth the nodes of a mesh so that the fewest elements have a target metric value outside of the range (i.e. they *fail*). An alternative objective could be to minimise (or maximise) the highest (or lowest) target element quality metric value. Constraints can also be added to keep other *constraint* element quality metrics within their specified ranges.

An element quality metric is a function that evaluates the node positions of an element to give a value. Many are routinely used for assessing mesh quality such as skew, warp, jacobian etc. [1]. Often their response surface plots have discontinuous features which can cause problems in gradient based optimisation methods.

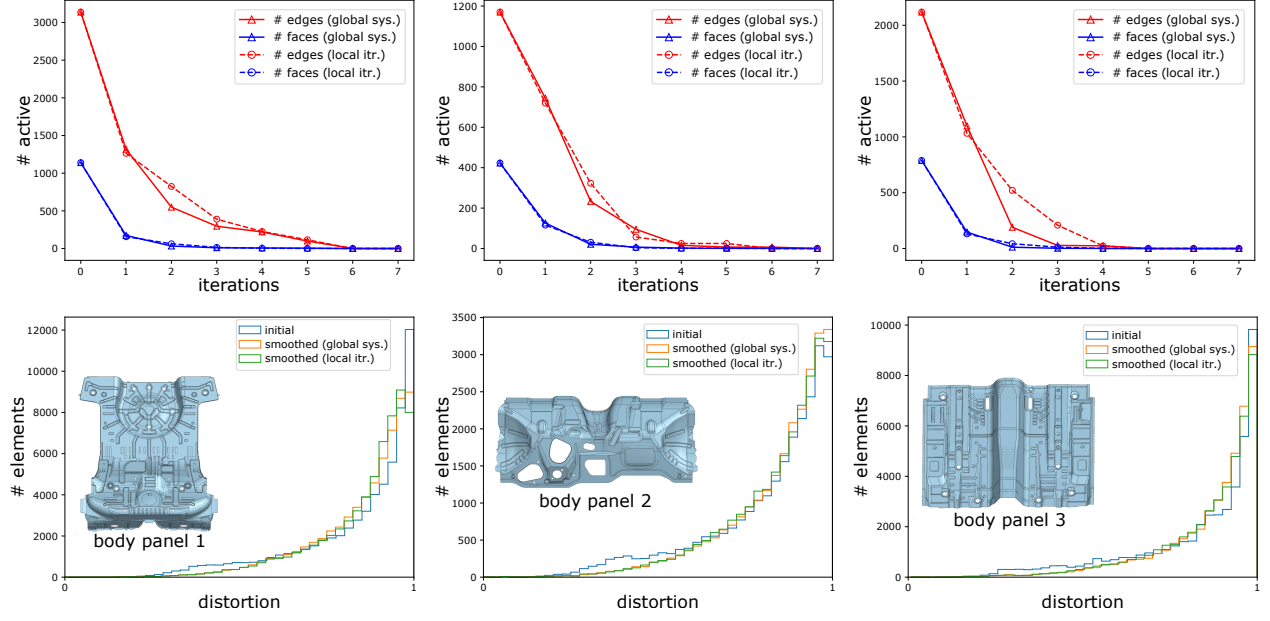
Taper is an element quality metric that often sees many failures as it is not tightly coupled with more intuitive element quality metrics such as min/max angles and aspect ratio. Using the NX Nastran definition [1] (other slightly different formulae exist), the taper of a quad element  $ABCD$  is computed by

$$\text{taper} = \frac{A_{\text{max}} - Q}{Q}, \quad (15)$$

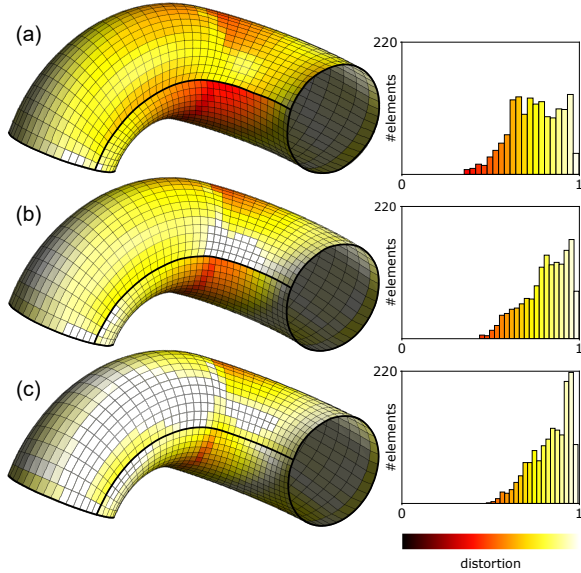
where  $A_{\text{max}}$  is the largest of the triangle areas  $ABD$ ,  $BCA$ ,  $CBD$  and  $DAC$  and  $Q = 0.5 \times \text{total quad area}$ . It does not apply to triangle elements. The ideal taper value is 0.0 and the worst value is 1.0 so it should be minimised in optimisation. For certain structural



**Figure 7:** Global smoothing of a quad dominant shell mesh on an body panel.



**Figure 8:** Comparison between the global system and local iteration methods for three body panels. Graphs showing the number of unconverged (active) edges and faces per iteration for global system and local iteration methods (top). Histograms showing mesh distortion improvements (bottom).



**Figure 9:** Smoothing a quad mesh on a pipe model. (a) Initial mesh. (b) Smoothed mesh using local iteration scheme. (c) Smoothed mesh using global system scheme. The histograms show the distributions of element distortion.

mechanics analyses it is required that the taper values of all quad elements are less than 0.5.

Local optimisation smoothing is performed in a iteration scheme. The whole process is outlined in algo-

rithm 2.

#### Algorithm 2: Local optimisation smoother

```

for i in range(0, n_max):
    get failing elements
    if no failing elements:
        break
    for node in failing elements:
        optimise node position

```

First all the elements that fail the target quality metric are identified. Next all the positions of the nodes of the failing elements are optimised with respect to their one-rings (adjacent elements and nodes). Nodes on faces are visited first before nodes on edges. This is done repeatedly for a fixed number of iterations or until there are no failing elements.

Algorithm 3 describes the process for node optimisation.

#### Algorithm 3: Optimisation of a node position

```

if node on edge:
    get previous and next nodes
    locally parametrise edge (node position
    → = pos(t))
    optimise t
    if t is valid:
        update node position
else: #node on face
    locally parametrise one ring (node
    → position = pos(u,v))
    optimise (u,v)
    if (u,v) is valid:
        update node position

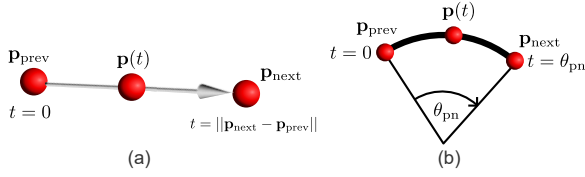
```

A node's position is optimised in a local parametric space that is established immediately before the optimisation process. For an interior node of a face



the parametrisation method that is used is an orthographic projection onto a local tangent plane. The angle-weighted normal of the one-ring elements is used for the normal. Minimal distortion is expected in this parametrisation as it is applied locally, except for extreme curvature cases. The search space of  $uv$  coordinates is the area of the projected one-ring elements.

Nodes on edges are parametrised locally in one of two ways with the choice depending on the relative positions of the node, its previous node and its next node along the edge. If the positions are almost co-linear then the edge is parametrised as a straight line, as shown in Fig. 10 (a). In this case the search space is  $t \in (0, \|\mathbf{p}_{\text{next}} - \mathbf{p}_{\text{prev}}\|)$ . Otherwise a circle is fitted to



**Figure 10:** Local edge parametrisation. (a) Straight line parametrisation; (b) Arc parametrisation.

the three nodes and the parametric value corresponds to a subtended angle as shown in Fig. 10 (b). In this case the search space is  $t \in (0, \theta_{pn})$ .

The default objective function that is used is

$$f = \sum_{i=1}^{\# \text{ one-ring elements}} \text{target\_metric}(\text{element}_i)^2. \quad (16)$$

Through testing this has been found to be the most effective objective function for minimising the number of failing elements. If the overall goal is to to minimise (or maximise) the highest (or lowest) target element quality metric value then tests indicated that it is better to do all iterations using equation (16) as the objective function except for the final iteration for which the following objective function is used,

$$f = \min(\text{target\_metric}(\text{element}_i)) \quad | \ i \in [1, \# \text{ one-ring elements}]. \quad (17)$$

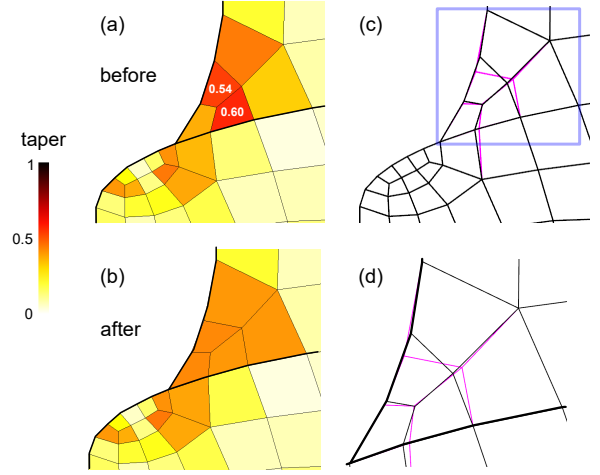
Powell's method [17] is used as the optimisation method. No derivatives of the objective function are taken, which is convenient because the objective functions are typically not fully differentiable. However, it may not be as efficient as other non-linear optimisation methods that use derivatives. The initial node position is used as the initial starting point. Constraints are added to the optimisation method which are of the form:

$$\text{constraint\_metric}(\text{element}) < \text{threshold}. \quad (18)$$

For example, a constraint on the maximum corner angle of an element to be less than  $150^\circ$ .

## 5.1 Results

In Fig. 11 an example is shown of a close up region of a quad mesh between two connected faces. The local optimisation smoother was used to resolve two elements failing taper, i.e. elements with a taper element quality metric value greater than 0.5. The nodes of the elements were perturbed along the faces and along edges to improve taper.



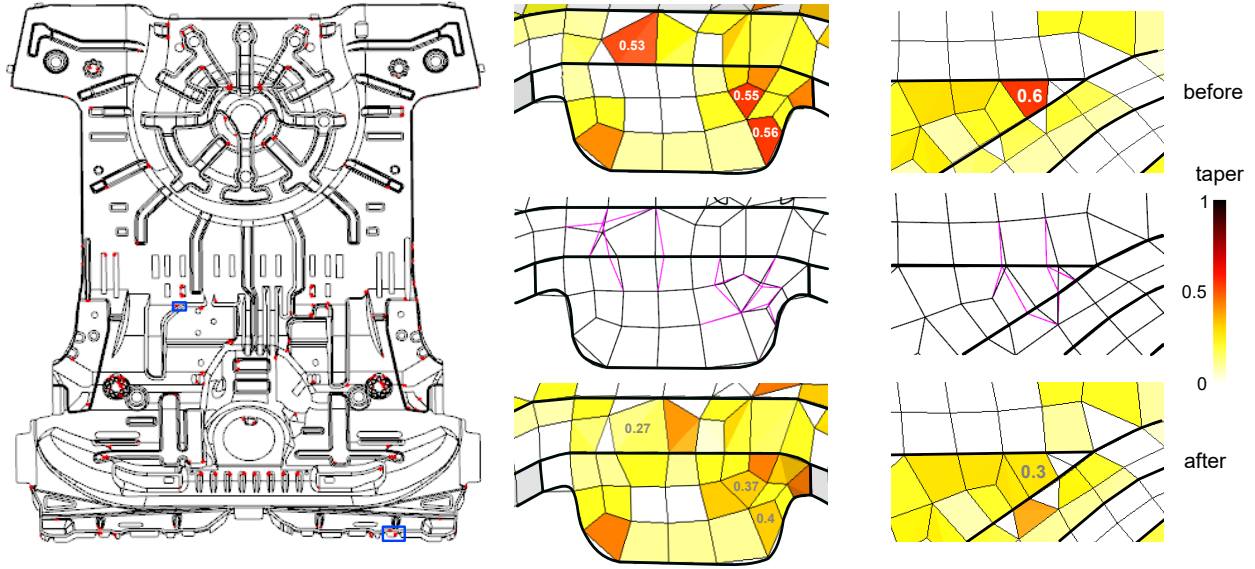
**Figure 11:** An example of optimisation smoothing for fixing taper failures ( $> 0.5$ ). (a) The initial mesh with two failures; (b) The smoothed mesh with no failures; (c & d) Overlays of the meshes (initial mesh in black and smoothed mesh in magenta).

In Fig. 12 the local optimisation smoother was used to resolve taper failures in the quad dominant mesh of a body panel to which the global smoother was previously applied (Fig. 7). Three constraints are applied to keep the minimum and maximum corner angles and the minimum edge lengths of the elements within acceptable ranges. Twelve iterations are performed and the taper failures are reduced from 179 to 47, as shown in Fig. 13. Noticeably, most of the improvement is achieved in the first iteration and thereafter the convergence is slow. The timings are reported in Tab. 3. The mapping and inverse mapping times are negligible and most of the time is taken in the Powell optimisation solver, which typically converges in less than 5 iterations for each optimisation. The whole process is reasonably quick (3.2s) which demonstrates its effectiveness on a complex geometry.

## 6. LIMITATIONS

The global smoother has two theoretical limitations. Firstly, the variational principle is inconsistent for the separate stages of smoothing on edges and faces. Nodes on edges are smoothed by Laplacian smoothing (Eqn. (1)) in 3D whereas nodes on faces are smoothed according to another variational principle (Eqn. (9)) in

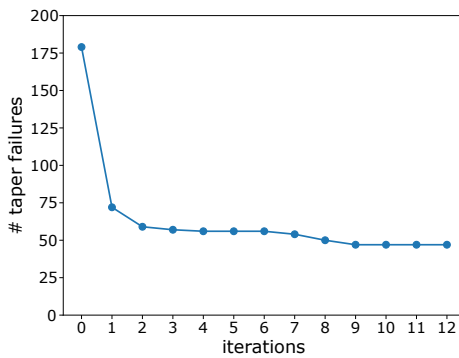




**Figure 12:** Optimisation smoothing of the quad dominant body panel mesh to resolve taper failures. To the left the whole body panel is shown with the the failing elements highlighted in red. To the right close ups of two regions are shown that demonstrate how the nodes are perturbed to resolve the failures.

Target metric	<i>Taper</i>
Constraint metrics	<i>Min Angle, Max Angle, Min Edge Length</i>
Num. edge optimisations	1621
Num. face optimisations	387
Edge optimisation time	0.712s
Face optimisation time	2.518s
Initialisation time	0.008s
Mapping time	<0.001s
Inverse mapping time	0.002s
Total time	3.240s

**Table 3:** Timings for the local optimisation smoother on the body panel mesh.



**Figure 13:** Number of elements failing Taper versus iterations in the optimisation smoother for the quad dominant body panel mesh.

2D. From a theoretical standpoint it may be preferable in edge smoothing to minimise the sum squared of the

variational principles of the connected faces. However, this formulation would be more complex and less well behaved than the linear system of the employed formulation. In practice this inconsistency has not been found to be an issue and convergence is achieved in all tests within a few iterations.

Secondly, smoothing on faces is performed in 2D parametric space but the quality of the mesh is measured in 3D. Therefore any distortion of the parametrisation compromises the effectiveness of the smoothing. But typical parametrisations of faceted B-rep faces do not exhibit severe distortion and this limitation is not significant. Other authors have developed smoothing methods that can account for the parametric space distortion [2, 18]. But incorporating these adjustments would lead to non-linear equations which would be more difficult to solve. Also, they involve parametric derivatives which are not directly available in our case.

A practical limitation is that a varying target element size cannot be accommodated. This would be possible in the edge smoothing method by adding a penalty term to Eqn. (8) for deviation of edge length from the target element size. However, the system would then become non-linear so an iterative line-search solver would have to be used. Varying target element size could also be accommodated be in face variational smoothing by adding space-weighting terms to Eqn. (14). Implementing these will be the subject of future work.

## 7. CONCLUSION

In this work we have described a process for smoothing shell meshes on 3D faceted B-rep geometries. The approach comprises a two-stage method whereby the mesh is first globally smoothed for large-scale movement of nodes across faces and along edges to minimise a variational principle. Finally, the mesh is fined-tuned using a local optimisation method to improve element quality metrics which may have non-smooth response surfaces and hence are not suitable for use in global objective functions. Constraints can be added to the local optimisation method to ensure that key element metrics are not regressed.

The global smoothing procedure involves iteratively smoothing nodes on edges and then smoothing nodes on faces. Smoothing on edges is done in 3D with tangent line constraints which avoids the need to parametrise the edges. Complex edges with  $G^1$  discontinuities can be robustly handled with this approach. Although smoothing nodes on edges not a completely new concept it has not been widely adopted especially on faceted geometries. The smoothing on faces is done in parametric space and a variational method is used. The variational principle that is used may be adjusted to suit the mesh properties most desired. A weighted sum of the Length Squared, Area Squared, Orthogonality and Winslow principles with well chosen weights will tend to produce good quality inversion-free meshes.

Results indicate that both smoothers perform effectively on complex models with reasonable execution times. Both a local iteration scheme and a global system scheme have been tested in global smoothing. It has been found that the global system scheme can produce superior results when the nodes must travel relatively large distances along the faces. However, there is a theoretical risk that elements may be inverted although in practice this has not been observed. The advantage of the local iteration scheme is that checks can be easily performed after each local smoothing operation to ensure that certain element properties are preserved. For example it is important that element edge length is kept above a specified tolerance for transient dynamic analyses.

The combination of both smoothers gives a comprehensive process for generally improving the mesh quality whilst also targeting particular element quality metrics. The methods have been implemented in the commercial CAE software package Siemens Simcenter 3D.

## References

- [1] "NX Nastran User's Guide." [https://docs.plm.automation.siemens.com/data\\_services/](https://docs.plm.automation.siemens.com/data_services/resources/nxnastran/11/help/tdoc/en_US/pdf/User.pdf)

- resources/nxnastran/11/help/tdoc/en\_US/pdf/User.pdf. Accessed: 18-Jan-2022
- [2] Ruiz-Gironés E., Roca X., Sarrate J. "Optimizing Mesh Distortion by Hierarchical Iteration Relocation of the Nodes on the CAD Entities." *Procedia Engineering*, vol. 82, 101–113, 2014. 23rd International Meshing Roundtable (IMR23)
- [3] Garimella R.V., Shashkov M.J. "Polygonal surface mesh optimization." *Engineering with Computers*, vol. 20, no. 3, 265–272, Sep 2004
- [4] Garimella R.V., Shashkov M.J., Knupp P.M. "Triangular and quadrilateral surface mesh quality optimization using local parametrization." *Computer Methods in Applied Mechanics and Engineering*, vol. 193, no. 9, 913–928, 2004
- [5] Shivanna K.H., Grosland N., Magnotta V. "An Analytical Framework for Quadrilateral Surface Mesh Improvement with an Underlying Triangulated Surface Definition." *Proceedings of the 19th International Meshing Roundtable*. 2010
- [6] Escobar J.M., Montero G., Montenegro R., Rodríguez E. "An algebraic method for smoothing surface triangulations on a local parametric space." *International Journal for Numerical Methods in Engineering*, vol. 66, 740–760, 2006
- [7] Gargallo-Peiró A., Roca X., Sarrate J. "A surface mesh smoothing and untangling method independent of the CAD parameterization." *Computational Mechanics*, vol. 53, 587–609, 2013
- [8] "Siemens Simcenter 3D." <https://www.plm.automation.siemens.com/global/en/products/simcenter/simcenter-3d.html>. Accessed: 21-Sep-2021
- [9] Hormann K., Polthier K., Sheffer A. "Mesh Parameterization: Theory and Practice." *ACM SIGGRAPH ASIA 2008 Courses*, SIGGRAPH Asia '08, pp. 12:1–12:87. ACM, New York, NY, USA, 2008
- [10] Jacobson A., Panozzo D., et al. "libigl: A simple C++ geometry processing library." <https://libigl.github.io/libigl-python-bindings/tut-chapter4/>, <https://libigl.github.io/>, 2018. Accessed: 28-Sep-2021
- [11] Lee C., Lo S. "A new scheme for the generation of a graded quadrilateral mesh." *Computers & Structures*, vol. 52, no. 5, 847–857, 1994
- [12] Knupp P.M. "Winslow Smoothing on Two-Dimensional Unstructured Meshes." *Proceedings of the 7th International Meshing Roundtable, Park City, UT*, pp. 449–457. 1998
- [13] Knupp P. *The fundamentals of grid generation*. CRC Press, Boca Raton, 1993
- [14] Bracci M., Tarini M., Pietroni N., Livesu M., Cignoni P. "HexaLab.net: An online viewer for hexahedral meshes." *Computer-Aided Design*, vol. 110, 24–36, 2019
- [15] "Opel international." <https://www.opel.com/>. Accessed: 14-Oct-2021
- [16] Guennebaud G., Jacob B., et al. "Eigen v3." <http://eigen.tuxfamily.org>, 2010
- [17] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. *Chapter 10. Minimization or Maximization of Functions, Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, USA, 3 edn., 2007
- [18] Gargallo-Peiró A., Roca X., Peraire J., Sarrate J. "A distortion measure to validate and generate curved high-order meshes on CAD surfaces with independence of parameterization." *International Journal for Numerical Methods in Engineering*, vol. 106, 1100–1130, 2016